Neural Operator for Scientific Computing

Thesis by
Zongyi Li

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2025
Defended May 9th, 2025

Zongyi Li
ORCID: 0000-0003-2081-9665

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Anima Anandkumar, for her groundbreaking research vision, unwavering support, and encouragement. Thank you for introducing me to this field since my undergraduate years, and most importantly, teaching me the precious courage to pursue the difficult path. I am equally indebted to Andrew Stuart for being an inspiring mentor whose vast knowledge and incredible kindness have helped me grow both academically and personally. I extend my sincere appreciation to the members of my thesis committee, Kaushik Bhattacharya, Thomas Yizhao Hou, Oscar Bruno, and Pedram Hassanzadeh, for teaching me applied mathematics, collaborating with me, and supporting me throughout my doctoral journey.

Special thanks go to my close collaborators, Nikola Kovachki and Samuel Lanthaler, who have been like second advisors to me. I have learned immensely from their expertise and approach to research. I am grateful to my early mentors, Kamyar Azizzadenasheli, Rose Yu, and Jean Kossaifi, for their guidance during the formative years of my PhD. I want to thank my senior collaborators: Tapio Schneider, Gianmarco Pinton, H. Jane Bae, Arvind Ramanathan, Morteza Gharib, Chiara Daraio who greatly extended my vision into science. I also want to thank my brilliant collaborators Burigede Liu, Daniel Zhengyu Huang, Yuanyuan Shi, Gege Wen, Vignesh Gopakumar, Di Zhou, Daniel Leibovici, and Yixuan Wang. It has been a privilege to work alongside such talented researchers. In addition, I am thankful to Karthik Duraisamy, Paris Perdikaris, Andrej Risteski, Johannes Brandstetter, Haoxing Ren, and Sanjay Choudhry for helping me.

My academic journey has been enriched by the camaraderie of an amazing community: Sahin Lale, Florian Schaefer, Jiawei Zhao, Yujia Huang, Hongkai Zheng, Guanzhi Wang, Chuwei Wang, Robert Joseph George, Beom Seok Kang, David Pitt, and Jiachen Yao. I am equally thankful for the postdoctoral members of our group: Anqi Liu, Pan Xu, Huaizu Jiang, Rafal Kocielnik, Peter Jiayun Wang, Julius Berner, Kaiyu Yang, Bahareh Tolooshams, Valentin Duruisseaux, Colin White, and Luo Cheng. I have been fortunate to work with exceptional undergraduate students: David Jin, Derek Qin, Miguel Liu-Schiaffini, Kimia Hassibi, Haydn Maust, Zelin Zhao, Catherine Deng, Vansh Tibrewal, Xinyi Li, Reva Dhillon, and Michael Chen. Together, we have formed an inspiring academic family whose companionship I have truly cherished.

I am grateful to Yisong Yue, Adam Wierman, Katie Bouman, and Georgia Gkioxari for fostering an inclusive and vibrant departmental environment. I would like to thank my fellow graduate students: Jiajie Chen, Yifan Chen, Berthy Feng, Angela Gao, Ivan Jimenez Rodriguez, Dohyeon Kim, Matthew Levine, Ziqi Ma, Nicholas Nelsen, Manuel Santana, Sabera Talukder, and Siki Wang. They have created a wonderful community that has supported me throughout this journey. Special thanks to my friends Xin Tong, Yue Liu, Han Liu, Shuming Zhang, Ran Gao, Zihui Wu, Jiaye Wu, Guandao Yang, and Tanya Marwah for the enjoyable moments and fun times we have shared. I also extend my thanks to the outstanding administrative staff in the CMS department and at Caltech, especially Alma Rangel-Fuentes, Maria Lopez, Jolene Brink, and Carmen Nemer-Sirois. It has truly been a privilege to be a student at Caltech.

Finally, I want to express my profound gratitude to my father, Wen Li, my mother, Yong Lin, my brother, Jiayi Li, and my grandmother, Wenshu Peng, for their unconditional love and trust. Thank you for inspiring me since an early age and encouraging me to pursue my own path. I am deeply thankful to my partner, Jingyi Qiu, for her compassion, support, and companionship, especially during difficult times. Thank you for bringing me joy, exploring the world with me, and growing together. This thesis is dedicated to you all. Thank you!

ABSTRACT

Scientific computing, which aims to accurately simulate complex physical phenomena, often requires substantial computational resources. By viewing data as continuous functions, we leverage the smoothness structures of function spaces to enable efficient large-scale simulations. We introduce the neural operator, a universal machine learning framework designed to approximate solution operators in infinite-dimensional spaces, achieving scalable physical simulations. The thesis begins with the introduction and definition of neural operators. Chapters 2-4 discuss architecture designs of neural operators including graph neural operator, multipole neural operator, and Fourier neural operator. Chapters 5-7 discuss physics-based learning techniques such as dissipative loss, physics-informed loss, and scale consistency loss. Chapters 8-10 discuss geometric neural operators with various boundary shapes, including latent space embedding, learned deformation, and optimal transport. Chapters 11-12 discuss further applications of neural operator in weather forecast and carbon capture & storage.

PUBLISHED CONTENT AND CONTRIBUTIONS

Li, Zongyi, Daniel Zhengyu Huang, et al. (2023). "Fourier neural operator with learned deformations for pdes on general geometries". In: *Journal of Machine Learning Research* 24.388, pp. 1–26.
Z.L. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and wrote the majority of the manuscript.

Li, Zongyi, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, et al. (2020). "Neural operator: Graph kernel network for partial differential equations". In: URL: `https://arxiv.org/abs/2003.03485`.
Z.L. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and wrote the majority of the manuscript.

– (2021). "Fourier neural operator for parametric partial differential equations". In: *International Conference on Learning Representations*.
Z.L. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and wrote the majority of the manuscript.

Li, Zongyi, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, et al. (2020). "Multipole graph neural operator for parametric partial differential equations". In: *Advances in Neural Information Processing Systems (Neurips)* 33, pp. 6755–6766.
Z.L. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and wrote the majority of the manuscript.

Li, Zongyi, Nikola Kovachki, Chris Choy, et al. (2023). "Geometry-informed neural operator for large-scale 3D PDEs". In: *Advances in Neural Information Processing Systems (Neurips)* 36.
Z.L. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and wrote parts of manuscript.

Li, Zongyi, Miguel Liu-Schiaffini, et al. (2022). "Learning Dissipative Dynamics in Chaotic Systems". In: *Advances in Neural Information Processing Systems (Neurips)*.
Z.L. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and wrote the majority of the manuscript.

Li, Zongyi, Hongkai Zheng, et al. (2024). "Physics-informed neural operator for learning partial differential equations". In: *ACM/JMS Journal of Data Science* 1.3, pp. 1–27.

Z.L. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and wrote the majority of the manuscript.

Pathak, Jaideep et al. (2023). "Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators". In: *PASC '23: Proceedings of the Platform for Advanced Scientific Computing Conference*.
Z.L. participated in designing the model and methodology.

Wen, Gege et al. (2023). "Real-time high-resolution CO 2 geological storage prediction using nested Fourier neural operators". In: *Energy & Environmental Science* 16.4, pp. 1732–1741.
Z.L. participated in designing the model and methodology.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Machine Learning for Scientific Computing

Advancements in machine learning (ML) are driving paradigm shifts in the field of scientific computing. Traditionally, many scientific applications rely on brute-force numerical methods using high-performance computing (HPC). Despite increasing computational power, these traditional approaches face significant speed and scalability limitations for complex systems. For instance, simulations of turbulent flows or climate systems can take weeks on supercomputers, hindering scientific progress. This thesis aims to address these challenges by leveraging ML to accelerate and improve complex scientific simulations.

To address these limitations, researchers have been exploring data-driven surrogate models, particularly neural networks. However, standard neural networks, such as convolutional neural networks and transformers, were initially designed for machine learning tasks with image data and language tokens. The challenge lies in adapting these AI methods for scientific computing, where data typically consists of functions which are high-resolution, multi-scale, with complex geometries, and must adhere to physical principles. Direct use of standard deep learning models without careful adaptation or physical constraints may lead to ill-definedness and inconsistency (Kovachki et al., 2021). This raises key questions: How can AI augment or replace traditional brute-force calculations? What is the most general yet mathematically well-defined model architecture for physical systems? And can we eventually extrapolate beyond the training data to find unique designs and discover new science? Answering these questions has the potential to lead to paradigm shifts in scientific research.

Continuum Mechanics. Physical phenomena can be classified by scale: quantum, microscopic, mesoscopic, and macroscopic. In recent years, massive progress has been made in microscale simulation, especially molecular dynamics for drug discovery (Jumper et al., 2021; Baek et al., 2021). This progress is largely enabled by machine learning models that can learn and predict complex molecular interactions or force fields with high accuracy, significantly accelerating the simulation process. While architectures like the transformer (Vaswani et al., 2017) have proven powerful

in related biomolecular tasks like protein structure prediction, the challenge of accurately and efficiently simulating the dynamics of large numbers of atoms at scale remains.

However, larger scale physics and mechanics, such as fluid mechanics, solid mechanics, and climate science, present significant computational and modeling challenges. For macroscopic or mesoscopic scale problems, the domain is so large that explicitly modeling every discrete particle is computationally prohibitive. Instead, the continuum assumption is made, treating the material as continuous where nearby particles are similar. Therefore, instead of studying particle-based representations, function-based representations are used. For continuum mechanics, the physical interactions can often be described by partial differential equations (PDEs).

Partial Differential Equations Consider a leading example: the Poisson-type second order elliptic equation (Darcy Flow) that describes steady state diffusion of heat or pressure:

$$\begin{aligned} -\nabla \cdot (a \nabla u) &= f \quad \Omega \\ u &= g \quad \partial\Omega \end{aligned} \tag{1.1}$$

Here, a is the diffusion coefficient or heat conductivity; u is the pressure field or the temperature field; f is the forcing function; and g is the boundary condition. A standard forward problem is to find the corresponding solution function u, given the coefficient function a, forcing f, and boundary g. Conversely, the inverse problem is to find a or f given u.

In the past fifty years, massive progress has been made on numerical solvers for partial differential equations, such as finite difference methods, finite element methods, and spectral methods. These methods discretize the domains or function spaces and then solve discrete linear or non-linear problems. Researchers have developed numerical methods with excellent convergence rates and stability guarantees. However, two significant challenges remain, particularly for complex, large-scale systems.

First, standard numerical simulations rely on mathematical models in the form of partial differential equations, which inherently introduce aspects of simplification and approximation. For example, while PDEs like the Navier-Stokes equation are powerful models, they are not always a perfect representation of all aspects of real-world fluid motions under all conditions. Mathematically, fundamental questions remain, such as the existence of smooth solutions for the 3D incompressible Navier-

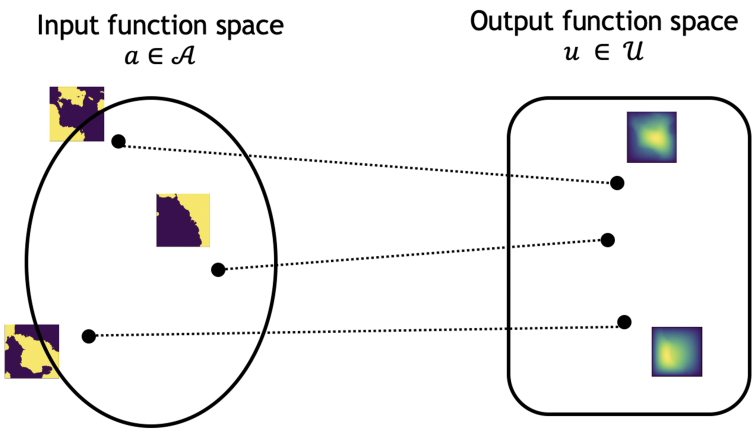


Figure 1.1: Neural operator: mapping between function spaces

Stokes equations, with some evidence regarding related equations like the Euler equation suggesting potential blowups (Chen and Hou, 2022). Physically, viscous dissipation in the Navier-Stokes equation inherently involves energy transformation, often resulting in thermal effects. For accurate modeling of many complex real-world flows, particularly those with significant temperature variations or compressibility, a coupled set of equations, including an energy equation, is necessary to capture all relevant physics.

Furthermore, standard numerical simulations can be computationally prohibitive for large-scale problems. Numerical methods require a sufficiently fine discretization that can resolve the differential operators and relevant physical scales. While methods relying on convergence ($h \mapsto 0$) work very well for smaller scale and lower dimensional problems, Direct Numerical Simulation (DNS) of large-scale, high-Reynolds number flows is usually computationally unaffordable. To address this, researchers have studied sub-grid models such as the Reynolds Averaged Navier Stokes Equation (RANS) or Large Eddy Simulation (LES) with turbulence models. Such models usually introduce further approximations with empirical parameters often found in experiments. There has been ongoing debate on the universal validity and accuracy of these subgrid models for all flow regimes (Pope, 2004). The computational cost associated with fine discretization has been an obstacle for larger scale, high-resolution numerical simulations such as weather forecasting and 3D aerodynamics design. Machine learning models offer the hope to complement or go beyond traditional numerical simulations, potentially by learning directly from or being constrained by real-world observational data.

1.2 Neural Operator: Mapping Between Function Spaces

This research focuses on developing machine learning approaches that effectively handle the complex, smooth structures inherent in scientific computing data. A central theme of this work is the development of operator learning methods, particularly neural operators, for solving partial differential equations (PDEs). For the example of Equation 1.1, the neural operator aim to approximation the solution operator mapping from the space of coefficient $\mathcal{A} = \mathcal{L}^\infty$ to the space of solution $\mathcal{U} = \mathcal{H}^1$, as shown in Figure 1.1.

$$\mathcal{G} : a \mapsto u$$

Neural operators are a generalization of neural networks designed to learn mappings between infinite-dimensional function spaces, from input coefficients (or initial conditions) to output solutions. The neural operator is formulated as a composition of linear integral operators $\mathcal{K}$ with nonlinear functions f (channel mixing), augmented with lifting (encoder) $\mathcal{P}$ and projection (decoder) $\mathcal{Q}$. Neural operators are discretization-convergent, i.e., they share the same model parameters across different discretizations and converge to the target continuum operator as the discretization refines.

$$\mathcal{G} = \mathcal{Q} \circ (f_l \circ \mathcal{K}_l) \circ \ldots \circ (f_1 \circ \mathcal{K}_1) \circ \mathcal{P}$$

Earlier works implemented the integral operators as message passing on the graph, leading to Graph Neural Operator (GNO) (Z. Li, Kovachki, Azizzadenesheli, Liu, Bhattacharya, et al., 2020) and its enhancement based on the Multi-pole method (MGNO) (Z. Li, Kovachki, Azizzadenesheli, Liu, A. Stuart, et al., 2020). These methods are highly flexible and they showed promising results on smaller-scale PDEs; however, the graph implementation is relatively slow due to its local connections. To efficiently capture long-range interactions, the Fourier Neural Operator (FNO), which has been a key contribution in this field, was proposed.

$$(\mathcal{K}v)(x) : \int \kappa(x, y)v(y)\mathrm{d}y$$

Fourier Neural Operator. Building on the concept of neural operators, the Fourier Neural Operator (FNO) was introduced (Z. Li, Kovachki, Azizzadenesheli, Liu, Bhattacharya, et al., 2021). FNO parameterizes the integral kernel directly in Fourier space while keeping nonlinear activations in the spatial domain. Utilizing the Fast Fourier Transform (F), FNO scales quasi-linearly with the resolution, which offers higher efficiency compared to previous graph-based implementations.

$$(\mathcal{K}v)(x) : F^{-1}(\hat{\kappa} \cdot Fv)$$

Since operations that are **local** in spectral or spatial space are **global** in the other space, this approach enables FNO to effectively perform global operations in both domains. FNO achieves state-of-the-art performance on several benchmark problems, including Burgers' equation, Darcy Flow, and the Navier-Stokes equations. Notably, it is the first machine learning-based method to successfully model turbulent flows with zero-shot super-resolution capabilities. This approach demonstrates up to three orders of magnitude speedup compared to traditional PDE solvers while maintaining superior accuracy. Furthermore, FNO's discretization-invariance allows it to generalize across different grid resolutions and geometries, which yields a substantial advantage over existing deep learning methods for PDEs. Several extensions based on FNO are further investigated such as the Physics-Informed Neural Operator (PINO) (Z. Li, Zheng, et al., 2024b), which employs equation loss, and the Adaptive Fourier Neural Operator (AFNO) (Guibas et al., 2021), which does not truncate to a specific Fourier mode but uses thresholding. These models have been successfully applied to various domains (Azizzadenesheli et al., 2024; Pathak et al., 2022; Liu et al., 2021).

Physics Informed Learning. While the FNO demonstrated significant improvements in solving PDEs, the need to incorporate **physical constraints** and address limitations in data availability was recognized. This led to the development of Physics-Informed Neural Operators (PINO) (Z. Li, Zheng, et al., 2024a), a novel approach that bridges the gap between physics-informed optimization and data-driven neural operator learning for solving PDEs. PINO addresses key limitations of existing methods: (1) Physics-Informed Neural Networks (PINNs) often struggle with optimization challenges and fail to solve complex multi-scale dynamic systems. (2) Purely data-driven approaches like FNOs are limited by the availability and resolution of training data. To overcome these drawbacks, PINO combines training data with PDE constraints at different resolutions, allowing for high-fidelity reconstruction of the ground-truth operator. This hybrid approach enables PINO to learn accurate solution operators even with limited or low-resolution data, while also improving optimization stability. By incorporating PDE constraints at higher resolutions, PINO overcomes both the limitations of purely data-driven methods and the optimization challenges faced by PINNs.

Complex Geometry. One of the major challenges in scientific computing is handling complex 3D geometries, an area where standard FNO does not excel. While finite difference and pseudo-spectral methods work well on simple regular domains, meshing on complex geometries is extremely tedious, often requiring iterative mesh refinement. The goal was to develop a flexible method capable of handling arbitrary geometries through decomposition and deformation. To address this challenge, the Geometry-aware FNO (Geo-FNO) (Z. Li, Huang, et al., 2022), a novel framework for solving PDEs on arbitrary geometries, was developed. The core innovation lies in **learning a deformation** that maps the irregular physical domain to a uniform latent space where the FFT can be efficiently applied. The deformation induces **generalized harmonic series** on the manifold, which provides both efficiency and flexibility. For example, the cosine transform induces Chebyshev polynomials. This approach combines the computational efficiency of the FFT with the flexibility of learned deformations, resulting in a model that is as fast as FNO but more versatile and accurate for complex geometries. Geo-FNO's effectiveness was demonstrated on a variety of PDEs, including elasticity, plasticity, Euler's equation, and the Navier-Stokes equation, for both forward modeling and inverse design problems. Experiments yielded impressive results: Geo-FNO achieved up to 10^5 times acceleration compared to standard numerical solvers and halved the error rate relative to previous interpolation-based methods.

Attention Mechanism and Transformer Operator Interdisciplinary efforts often foster meaningful progress, which connect diverse fields and opening new pathways to address complex challenges. Building on neural operator approaches, concepts from other areas of machine learning, particularly the attention mechanism widely used in natural language processing, were explored. In (Guibas et al., 2021), Adaptive FNO (AFNO) was introduced, a novel approach to efficient token mixing in vision transformers. AFNO leverages the geometric structure of images to frame token mixing as a continuous global convolution, which enable high-resolution image processing without the quadratic complexity of self-attention. To adapt FNO for visual tasks, key modifications were made: a block-diagonal structure was imposed on channel mixing weights, adaptive weight sharing across tokens was implemented, and frequency modes were sparsified through soft-thresholding. AFNO achieves quasi-linear complexity in sequence length while maintaining high expressiveness. Experiments demonstrate AFNO's effectiveness across various tasks, including few-shot segmentation, high-resolution image segmentation, and

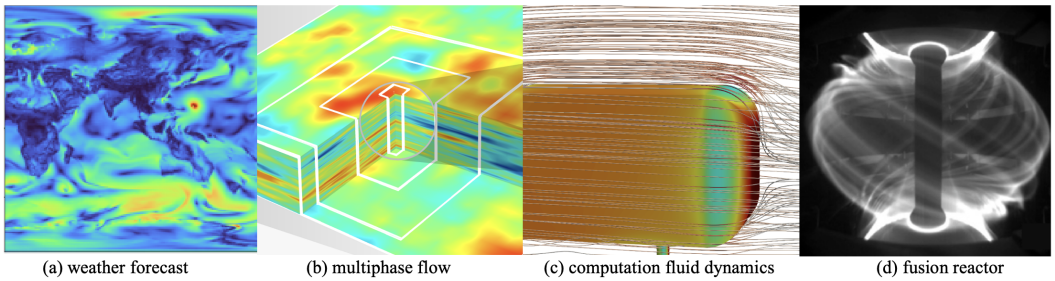


Figure 1.2: Applications of neural operators: (a) FourCastNet on the ERA5 weather prediction, (b) Multi-level FNO on 3D-time multi-phase flow, (c) Geometry-informed FNO on vehicle aerodynamics, (d) FNO on camera data of MAST reactor.

fluid dynamics simulations, where it outperforms self-attention in both efficiency and accuracy. as it outperforms self-attention in efficiency and accuracy for few-shot segmentation, high-resolution image segmentation, and fluid dynamics simulations.

Universal Approximation Theorem. As neural operator architectures became more sophisticated and demonstrated impressive empirical results, it is important to understand their theoretical foundations. This led to an investigation into the universal approximation capabilities of these neural operators. In (Kovachki et al., 2021), techniques from linear approximation are used to show that neural operators, in general, can approximate any continuous solution operators, including nonlinear ones, between pairs of arbitrary Sobolev spaces. Further, in (Lanthaler, Z. Li, and A. M. Stuart, 2023) the minimum necessary components for universal approximation are identified. It is demonstrated that combining nonlocality and nonlinearity in a simple way is sufficient to approximate any arbitrary continuous operator. A key contribution is the introduction of the Averaging Neural Operator (ANO), which reduces FNO to its core essence. It is proven that even this minimal ANO architecture, which utilizes only a spatial average as its nonlocal component, exhibits universal approximation capabilities. This finding challenges the prevailing view that an increasing number of Fourier modes is necessary for improved accuracy. This analysis unifies and extends universal approximation results across a wide range of neural operator architectures, including FNO, DeepONet, and other common models.

1.3 Applications in Science and Engineering

Research on neural operators has led to significant advancements in various scientific fields. By applying these techniques to real-world problems, the power and versatility of these approaches have been demonstrated:

Weather Forecast. Accurate and efficient weather forecasting is important for mitigating the impacts of extreme weather events, optimizing agricultural practices, and planning renewable energy resources. In collaboration with climate scientists and the Nvidia Earth-2 team, FNO's ability to handle complex, multi-scale phenomena was leveraged to develop FourCastNet (Pathak et al., 2022). This global data-driven weather forecasting model provides accurate short to medium-range global predictions at 0.25° resolution. FourCastNet excels in forecasting high-resolution, fast-timescale variables such as surface wind speed, precipitation, and atmospheric water vapor. Its performance matches or exceeds the standard ECMWF Integrated Forecasting System (IFS), particularly in small-scale features and precipitation. Remarkably, FourCastNet generates week-long forecasts in less than 2 seconds, approximately 45,000 times faster than traditional Numerical Weather Prediction (NWP) models. This speed facilitates the rapid creation of large-ensemble forecasts for improved probabilistic forecasting and uncertainty qualification. The model demonstrates exceptional performance in forecasting extreme events such as hurricanes and atmospheric rivers. It shows excellent capability in predicting near-surface wind speeds over land and coastal areas, which are critical for planning wind energy resource and managing extreme weather events. FourCastNet's impact on weather prediction mirrors AlexNet's revolutionary role in image classification. After validating this methodology, other leading tech companies like DeepMind and Microsoft built upon this work, helping establish a vibrant research field of weather forecast.

Carbon Capture and Storage. As global efforts to combat climate change grow, carbon capture and storage (CCS) in underground reservoirs has emerged as a critical technology for reducing atmospheric CO2 levels. Accurate simulation of CO2 migration is essential for ensuring the safety and efficiency of these storage systems. Extending the neural operator approach to multi-phase flow problems, innovative machine learning frameworks for high-resolution modeling of CO2 geological storage were developed (Wen, Z. Li, Azizzadenesheli, et al., 2022). The U-FNO model accurately predicts gas saturation and pressure buildup in complex CO2-water multi-phase flow scenarios, outperforming traditional CNN-based approaches in accuracy and data efficiency. Building on this, Nested FNO was introduced, which enables real-time, high-resolution 3D modeling at basin scale (Wen, Z. Li, Long, et al., 2023). Nested FNO produces forecasts nearly 700,000 times faster than existing methods, facilitating rapid probabilistic simulations that are crucial for scaling up

global CCS deployment. These models demonstrate superior performance in heterogeneous geological formations and critical applications such as determining gas saturation and pressure buildup fronts. By dramatically reducing computation time while maintaining high accuracy, these models provide invaluable tools for CCS project planning, risk assessment, and decision-making, enabling more comprehensive scenario analyses and risk evaluations crucial for accelerating global CCS deployment.

Computational Fluid Dynamics. Aerodynamic simulation plays an important role in optimizing the performance and efficiency of vehicles, aircraft, and other engineering systems, but traditional methods often struggle with high computational costs and complexity. For aerodynamic simulation on complex shapes, the Geometry-Informed Neural Operator (GINO) (Z. Li, Kovachki, Choy, et al., 2023) is introduced. GINO combines the strengths of GNO and FNO to handle irregular grids and capture global interactions efficiently. GINO was validated on two large-scale 3D computational fluid dynamics datasets, including industry-standard Ahmed's body geometries. Each dataset consists of 7M mesh points in the volume and 100k points on the surface. Results demonstrated that GINO achieves superior accuracy compared to existing methods, with error rates reduced by up to 25%. Moreover, GINO exhibited remarkable computational efficiency, accelerating drag coefficient calculations by a factor of 26,000 compared to the optimized GPU-based CFD simulators. This breakthrough in accuracy and speed positions GINO as a promising tool for rapid simulation and design optimization in complex engineering applications.

Nuclear Fusion. Nuclear Fusion, as the ultimate energy source, has been studied for more than 50 years. Controlling nuclear fusion requires accurate and efficient plasma modeling to avoid blow-up events. Collaborating with scientists from the UK Atomic Energy Authority, FNO was applied for modeling plasma evolution in both simulations and experiments (Gopakumar et al., 2024). The FNO model accurately solved the Magneto-Hydrodynamic (MHD) equations that describe plasma dynamics, achieving speeds 6 orders of magnitude faster than traditional numerical solvers while maintaining high accuracy. This approach was further extended to model plasma evolution observed by cameras in the MAST spherical tokamak, successfully forecasting filament formation and heat deposits. The camera-based FNO model could predict the full length of a plasma shot in half the actual shot

time. This work represents a significant advancement in real-time plasma modeling and prediction, with potential applications in plasma control and fusion reactor optimization.

1.4 Foundation Models

The long term goal of neural operator is to go beyond merely accelerating numerical solvers and focus on developing machine-learning models that can generalize outside their training data. While previous supervised machine learning approaches have shown success in scientific domains, generating the necessary training data often requires substantial computing resources for thousands of simulations, severely limiting its applicability. The diverse physical behaviors associated with different PDEs make it challenging to create a single universal model. Existing models focus on subsets of inputs, such as specific PDEs, but are often restricted to simple cases with one or two dimensions. This constraint leads to limited generalization, with models typically targeting only a single input variable while keeping other parameters fixed—unlike the flexibility of traditional numerical solvers. To address these limitations and amortize costs, the development of foundation models for scientific computing, trained on diverse datasets to capture a wide range of conditions, is proposed.

Multi-Scale Learning. Capturing multi-scale phenomena in scientific and engineering simulations remains one of the biggest challenges in computational modeling, particularly for complex systems like turbulence and lithography. Large-scale simulations in turbulence or lithography can involve billions of mesh points, reaching resolution of $10,000^2$ or $1,000^3$. Such multi-scale phenomena are difficult to capture accurately with single-scale models. To address this, the aim is to develop a multi-scale foundation model that can effectively handle a wide range of scales in PDEs. This approach is rooted in the principle of scale-consistency, a property inherent in many PDEs where solutions on larger domains, when restricted to subdomains, match solutions directly computed on those subdomains. To leverage this property, work is underway on a novel data augmentation scheme using sub-sampling and super-sampling techniques to bridge multi scales. This method creates virtual instances at various scales, potentially enhancing model generalization without requiring additional simulation data. This research has the potential to significantly improve the efficiency and generalizability of data-driven PDE solvers. Once generalization across different scales is achieved, the key question arises: Can

emergent behavior be found beyond the training scale? Specifically, if training data is available at the micro scale, can consistency be used to scale up and discover distinct behaviors at the macro scale such as hurricanes and blow-ups? By reducing the need for extensive training data and enabling more flexible, foundational models, this work could have far-reaching implications for a wide range of scientific and engineering applications, from fluid dynamics to electromagnetic simulations.

Multi-Fidelity Data. In scientific computing, high-quality data are often scarce. How can learning occur from a wide variety of data including low- and high-fidelity physical simulations, sparse observational and experimental data, graphics data, and existing physics knowledge? Current computational fluid dynamics (CFD) datasets for 3D simulations are especially resource-intensive to generate. Consequently, existing CFD datasets are limited to the order of thousands of cases, which is way lower than standard ML setting. To overcome this limitation, leveraging the abundance and accessibility of large-scale graphics datasets such as ShapeNet and Objverse is proposed. The approach involves pretraining a foundation model on diverse complex geometries from these graphics datasets, incorporating various representations, including Signed Distance Functions (SDFs) and solutions to the Poisson equation with both Dirichlet and Neumann boundary conditions. This strategy aims to create a versatile foundation capable of handling a wide array of geometric challenges. This model will then be fine-tuned by integrating fluid dynamics data from coarse simulations like RANS, LES, progressing to DNS and real experimental data. The resulting foundation model has the potential to not only accelerate simulations but also enable more efficient shape design and optimization in fields requiring both intricate geometries and sophisticated physics simulations, effectively bridging the gap between graphics and CFD.

AI-Aided Design. It is believed that machine learning methods will revolutionize the way design problems are approached. Design optimization problems require searching through a parameter space, which typically requires a large number of trials and runs. Due to excessive computational cost, traditional methods based on Bayesian methods and numerical solvers are usually limited to a low-dimensional parameter space. Especially in computational fluid dynamics, the standard industrial-level solvers based on RANS and LES omit all small-scale behaviors, designed in a highly empirical manner. In contrast, machine learning methods are fast, differentiable, and capable of handling high dimensional design space. The research

question is: Can AI discover novel designs and materials with unique geometry and structure? Initial works have shown neural operator can solve 2D inverse problem (Z. Li, Kovachki, Azizzadenesheli, Liu, Bhattacharya, et al., 2021), enhanced with physics constraint (Z. Li, Zheng, et al., 2024b) which show application in 2D airfoil (Z. Li, Kovachki, Choy, et al., 2023) and biomedical catheter design (Zhou et al., 2024). Combined with geometric methods, the aim is to further push design to 3D vehicle shapes and drone propellers. To go beyond numerical simulation used as training data, real-world experimental data can be used to augment the model (Renn et al., 2023). The goal is to discover new design shapes with less drag, and less noise, but higher resilience. In five to ten years, AI-based methods are expected to be as prevalent as standard methods such as RANS.

More is Different. While classical science based on reductionism attempts to break systems down into simpler parts and identify basic rules such as PDEs, this approach aims to develop machine learning-based models that capture complex global behaviors. AI-driven models have achieved several order of speedups in forecasting weather and simulating carbon storage (Pathak et al., 2022; Wen, Z. Li, Long, et al., 2023; Ye et al., 2024; Z. Li, Kovachki, Choy, et al., 2023). In near-term future, there has an ongoing progress towards a universal foundation model to handle a wide range of inputs such as initial conditions, coefficient conditions, domain sizes, geometries, and even multiple families of PDEs. As the model scales up to sufficient complexity, it will also reach unprecedented generalization. In long term, AI-driven methods are expected to find emergent behaviors in unseen scales and discover novel shapes and geometries for engineering design. Realizing such capabilities could lead to significant advancements in computational methods, greatly improving efficiency and accuracy in fields like rocket design and nuclear fusion. The realization of such a model would enable unprecedented efficiency and accuracy in computational methods, with far-reaching implications across multiple domains such as rocket design and nuclear fusion.

Future research directions converge towards addressing the fundamental challenges in applying neural operators to scientific computing: large scales, complex geometries, and extrapolation beyond the training regime. By developing increasingly flexible and compositional models, the aim is to progress towards a universal foundation model capable of solving a wide range of simulations with different initial condition, boundary condition, coefficient function, forcing function, domain size, geometry, and even multiple families of PDEs. The realization of such a model

would enable unprecedented efficiency and accuracy in computational methods, with far-reaching implications across multiple domains. For instance, it could accelerate climate modeling to enhance our understanding and mitigation of climate change, optimize spacecraft design for interplanetary exploration, and improve control systems for nuclear fusion reactors. Ultimately, this research seeks to push the boundaries of scientific computing, creating tools capable of tackling some of humanity's most challenging engineering problems and discovering new science.

1.5 Outline of the Thesis.

Chapters 2-4 discuss architecture designs of neural operators including graph neural operator, multipole neural operator, and Fourier neural operator. Chapters 5-7 discuss physics-based learning techniques such as dissipative loss, physics-informed loss, and scale consistency loss. Chapters 8-10 discuss geometric neural operators with various boundary shapes, including latent space embedding, learned deformation, and optimal transport. Chapters 11-12 discuss further applications of neural operator in weather forecast, carbon capture & storage.

References

Azizzadenesheli, Kamyar et al. (2024). "Neural operators for accelerating scientific simulations and design". In: *Nature Reviews Physics*, pp. 1–9.

Baek, Minkyung et al. (2021). "Accurate prediction of protein structures and interactions using a three-track neural network". In: *Science* 373.6557, pp. 871–876.

Chen, Jiajie and Thomas Y Hou (2022). "Stable nearly self-similar blowup of the 2D Boussinesq and 3D Euler equations with smooth data I: Analysis". In: *arXiv preprint arXiv:2210.07191*.

Gopakumar, Vignesh et al. (2024). "Fourier neural operator for plasma modelling". In: *Nuclear Fusion*.

Guibas, John et al. (2021). "Adaptive Fourier Neural Operators: Efficient Token Mixers for Transformers". In: *International Conference on Learning Representations, 2022*.

Jumper, John et al. (2021). "Highly accurate protein structure prediction with AlphaFold". In: *nature* 596.7873, pp. 583–589.

Kovachki, Nikola et al. (2021). "Neural operator: Learning maps between function spaces". In: *Journal of Machine Learning Research*, pp. 89–1.

Lanthaler, Samuel, Zongyi Li, and Andrew M Stuart (2023). "The nonlocal neural operator: Universal approximation". In: *arXiv preprint arXiv:2304.13221*.

Li, Zongyi, Daniel Zhengyu Huang, et al. (2022). "Fourier Neural Operator with Learned Deformations for PDEs on General Geometries". In: *To appear at JMLR*.

Li, Zongyi, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, et al. (2020). "Neural operator: Graph kernel network for partial differential equations". In: *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.

– (2021). "Fourier neural operator for parametric partial differential equations". In: *International Conference on Learning Representations*.

Li, Zongyi, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, et al. (2020). "Multipole graph neural operator for parametric partial differential equations". In: *Advances in Neural Information Processing Systems (Neurips)* 33, pp. 6755–6766.

Li, Zongyi, Nikola Kovachki, Chris Choy, et al. (2023). "Geometry-informed neural operator for large-scale 3D PDEs". In: *Advances in Neural Information Processing Systems (Neurips)* 36.

Li, Zongyi, Hongkai Zheng, et al. (2024a). "Physics-informed neural operator for learning partial differential equations". In: *ACM-JDS*.

– (2024b). "Physics-informed neural operator for learning partial differential equations". In: *ACM/JMS Journal of Data Science* 1.3, pp. 1–27.

Liu, Burigede et al. (2021). "A learning-based multiscale method and its application to inelastic impact problems". In: *Journal of the Mechanics and Physics of Solids*.

Pathak, Jaideep et al. (2022). "Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators". In: *arXiv preprint arXiv:2202.11214*.

Pope, Stephen B (2004). "Ten questions concerning the large-eddy simulation of turbulent flows". In: *New journal of Physics* 6.1, p. 35.

Renn, Peter I et al. (2023). "Forecasting subcritical cylinder wakes with Fourier Neural Operators". In: *arXiv preprint arXiv:2301.08290*.

Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc.

Wen, Gege, Zongyi Li, Kamyar Azizzadenesheli, et al. (2022). "U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow". In: *Advances in Water Resources* 163, p. 104180.

Wen, Gege, Zongyi Li, Qirui Long, et al. (2023). "Real-time high-resolution CO 2 geological storage prediction using nested Fourier neural operators". In: *Energy & Environmental Science* 16.4, pp. 1732–1741.

Ye, Teng et al. (2024). "Using Artificial Intelligence to Unlock Crowdfunding Success for Small Businesses". In: *arXiv preprint arXiv:2407.09480*.

Zhou, Tingtao et al. (2024). “AI-aided geometric design of anti-infection catheters”. In: *Science Advances* 10.1, eadj1741.

Chapter 2

MODEL: GRAPH NEURAL OPERATOR

The classical development of neural networks has been primarily for mappings between a finite-dimensional Euclidean space and a set of classes, or between two finite-dimensional Euclidean spaces. The purpose of this work is to generalize neural networks so that they can learn mappings between infinite-dimensional spaces (operators). The key innovation in our work is that a single set of network parameters, within a carefully designed network architecture, may be used to describe mappings between infinite-dimensional spaces *and* between different finite-dimensional approximations of those spaces. We formulate approximation of the infinite-dimensional mapping by composing nonlinear activation functions and a class of integral operators. The kernel integration is computed by message passing on graph networks. This approach has substantial practical consequences which we will illustrate in the context of mappings between input data to partial differential equations (PDEs) and their solutions. In this context, such learned networks can generalize among different approximation methods for the PDE (such as finite difference or finite element methods) and among approximations corresponding to different underlying levels of resolution and discretization. Experiments confirm that the proposed graph kernel network does have the desired properties and show competitive performance compared to the state of the art solvers.

2.1 Introduction

There are numerous applications in which it is desirable to learn a mapping between Banach spaces. In particular, either the input or the output space, or both, may be infinite-dimensional. The possibility of learning such mappings opens up a new class of problems in the design of neural networks, with widespread potential applicability. New ideas are required to build on traditional neural networks which are mappings from finite-dimensional Euclidean spaces into classes, or into another finite-dimensional Euclidean space. We study the development of neural networks in the setting in which the input and output spaces comprise real-valued functions defined on subsets in $\mathbb{R}^d$.

Related Work

We formulate a new class of neural networks, which are defined to map between spaces of functions on a bounded open set D in $\mathbb{R}^d$. Such neural networks, once trained, have the important property that they are discretization invariant, sharing the same network parameters between different discretizations. In contrast, standard neural network architectures depend heavily on the discretization and have difficulty in generalizing between different grid representations. Our methodology has an underlying Nyström approximation formulation (Nyström, 1930) which links different grids to a single set of network parameters. We illustrate the new conceptual class of neural networks within the context of partial differential equations, and the mapping between input data (in the form of a function) and output data (the function which solves the PDE). Both supervised and semisupervised settings are considered.

In PDE applications, the defining equations are often local, whilst the solution operator has non-local effects which, nonetheless, decay. Such non-local effects can be described by integral operators with graph approximations of Nyström type (Belongie et al., 2002) providing a consistent way of connecting different grid or data structures arising in computational methods. For this reason, graph networks hold great potential for the solution operators of PDEs, which is the point of departure for our work.

Partial differential equations (PDEs). A wide range of important engineering and physical problems are governed by PDEs. Over the past few decades, significant progress has been made on formulating (Gurtin, 1982) and solving (Johnson, 2012) the governing PDEs in many scientific fields from micro-scale problems (e.g., quantum and molecular dynamics) to macro-scale applications (e.g., civil and marine engineering). Despite the success in the application of PDEs to solve real-life problems, two significant challenges remain. First, identifying/formulating the underlying PDEs appropriate for the modeling of a specific problem usually requires extensive prior knowledge in the corresponding field which is then combined with universal conservation laws to design a predictive model; for example, modeling the deformation and fracture of solid structures requires detailed knowledge on the relationship between stress and strain in the constituent material. For complicated systems such as living cells, acquiring such knowledge is often elusive and formulating the governing PDE for these systems remains prohibitive; the possibility of learning such knowledge from data may revolutionize such fields. Second, solving complicated non-linear PDE systems (such as those arising in turbulence and plas-

ticity) is computationally demanding; again the possibility of using instances of data from such computations to design fast approximate solvers holds great potential. In both these challenges, if neural networks are to play a role in exploiting the increasing volume of available data, then there is a need to formulate them so that they are well-adapted to mappings between function spaces.

We first outline two major neural network based approaches for PDEs. We consider PDEs of the form

$$\begin{aligned} (\mathcal{L}_a u)(x) &= f(x), \qquad && x \in D \\ u(x) &= 0, && x \in \partial D, \end{aligned} \tag{2.1}$$

with solution $u : D \to \mathbb{R}$, and parameter $a : D \to \mathbb{R}$ entering the definition of $\mathcal{L}_a$. The domain D is discretized into K points (see Section 2.2) and N training pairs of coefficient functions and (approximate) solution functions $\{a_j, u_j\}_{j=1}^N$ are used to design a neural network. The first approach parametrizes the solution operator as a deep convolutional neural network between finite Euclidean space $\mathcal{G} : \mathbb{R}^K \times \Theta \to \mathbb{R}^K$ (Guo, Li, and Iorio, 2016; Zhu and Zabaras, 2018; Adler and Oktem, 2017; Bhatnagar et al., 2019). Such an approach is, by definition, not mesh independent and will need modifications to the architecture for different resolution and discretization of K in order to achieve consistent error (if at all possible). We demonstrate this issue in section 4.4 using the architecture of (Zhu and Zabaras, 2018) which was designed for the solution of (2.3) on a uniform 64×64 mesh. Furthermore, this approach is limited to the discretization size and geometry of the training data hence it is not possible to query solutions at new points in the domain. In contrast we show, for our method, both invariance of the error to grid resolution, and the ability to transfer the solution between meshes in section 4.4. Especially, (Ummenhofer et al., 2020) purposed continuous convolution network for fluid problems, where they sample points off grid and do linear interpolation on the grid, which may the convolution continuous compared to common CNN based method. However continuous convolution is still constrained by the underlying grid and preventing generalization of resolutions.

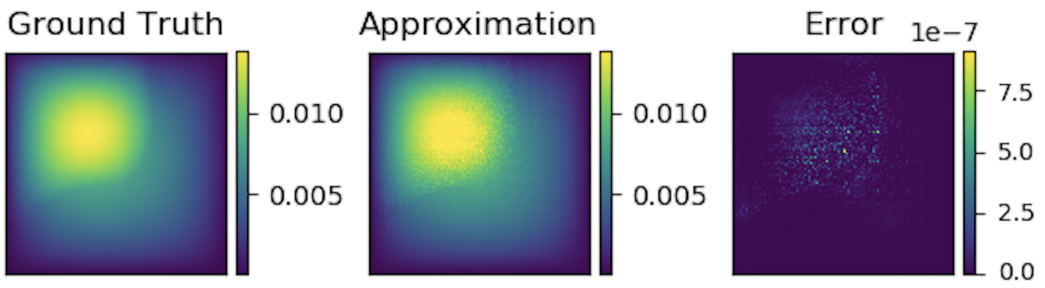
The second approach directly parameterizes the solution u as a neural network $\mathcal{G} : D \times \Theta \to \mathbb{R}$ (E and Yu, 2018; Raissi, Perdikaris, and Karniadakis, 2019; Bar and Sochen, 2019). This approach is, of course, mesh independent since the solution is defined on the physical domain. However, the parametric dependence is accounted for in a mesh-dependent fashion. Indeed, for any given new equation with new coefficient function a, one would need to train a new neural network $\mathcal{G}_a$. Such an

approach closely resembles classical methods such as finite elements, replacing the linear span of a finite set of local basis functions with the space of neural networks. This approach suffers from the same computational issue as the classical methods: one needs to solve an optimization problem for every new parameter. Furthermore, the approach is limited to a setting in which the underlying PDE is known; purely data-driven learning of a map between spaces of functions is not possible. The methodology we introduce circumvents these issues.

Our methodology most closely resembles the classical reduced basis method (R. A. DeVore, 2014) or the method of (Cohen and R. DeVore, 2015). Along with the contemporaneous work (Bhattacharya, Kovachki, and Stuart, 2020), our method, to the best of our knowledge, is the first practical deep learning method that is designed to learn maps between infinite-dimensional spaces. It remedies the mesh-dependent nature of the approach in (Guo, Li, and Iorio, 2016; Zhu and Zabaras, 2018; Adler and Oktem, 2017; Bhatnagar et al., 2019) by producing a quality of approximation that is invariant to the resolution of the function and it has the ability to transfer solutions between meshes. Moreover it need only be trained once on the equations set $\{a_j, u_j\}_{j=1}^N$; then, obtaining a solution for a new $a \sim \mu$, only requires a forward pass of the network, alleviating the major computational issues incurred in (E and Yu, 2018; Raissi, Perdikaris, and Karniadakis, 2019; Herrmann, Schwab, and Zech, 2020; Bar and Sochen, 2019). Lastly, our method requires no knowledge of the underlying PDE; the true map $\mathcal{G}^\dagger$ can be treated as a black-box, perhaps trained on experimental data or on the output of a costly computer simulation, not necessarily a PDE.

Graph neural networks. Graph neural network (GNNs), a class of neural networks that apply on graph-structured data, have recently been developed and seen a variety of applications. Graph networks incorporate an array of techniques such as graph convolution, edge convolution, attention, and graph pooling (Kipf and Welling, 2016; Hamilton, Ying, and Leskovec, 2017; Gilmer et al., 2017; Veličković et al., 2017; Murphy et al., 2018). GNNs have also been applied to the modeling of physical phenomena such as molecules (Chen et al., 2019) and rigid body systems (Battaglia et al., 2018), as these problems exhibit a natural graph interpretation: the particles are the nodes and the interactions are the edges.

The work (Alet et al., 2019) performed an initial study that employs graph networks on the problem of learning solutions to Poisson's equation among other physical



Graph kernel network for the solution of (2.3). It can be trained on a small resolution and will generalize to a large one. The Error is point-wise absolute squared error.

Figure 2.1: Graph Neural Operator trained on 16×16 and test on 241×241

applications. They propose an encoder-decoder setting, constructing graphs in the latent space and utilizing message passing between the encoder and decoder. However, their model uses a nearest neighbor structure that is unable to capture non-local dependencies as the mesh size is increased. In contrast, we directly construct a graph in which the nodes are located on the spatial domain of the output function. Through message passing, we are then able to directly learn the kernel of the network which approximates the PDE solution. When querying a new location, we simply add a new node to our spatial graph and connect it to the existing nodes, avoiding interpolation error by leveraging the power of the Nyström extension for integral operators.

Continuous neural networks. The concept of defining neural networks in infinite-dimensional spaces is a central problem that long been studied (Williams, 1996; Neal, 1996; Roux and Bengio, 2007; Globerson and Livni, 2016; Guss, 2016). The general idea is to take the infinite-width limit which yields a non-parametric method and has connections to Gaussian Process Regression (Neal, 1996; Matthews et al., 2018; Garriga-Alonso, Rasmussen, and Aitchison, 2018), leading to the introduction of deep Gaussian processes (**aretha**; Damianou and Lawrence, 2013). Thus far, such methods have not yielded efficient numerical algorithms that can parallel the success of convolutional or recurrent neural networks in finite dimensions in the setting of mappings between function spaces. Another idea is to simply define a sequence of compositions where each layer is a map between infinite-dimensional spaces with a finite-dimensional parametric dependence. This is the approach we take in this work, going a step further by sharing parameters between each layer.

Contributions

We introduce the concept of Neural Operator and instantiate it through *graph kernel networks*, a novel deep neural network method to learn the mapping between infinite-dimensional spaces of functions defined on bounded open subsets of $\mathbb{R}^d$.

- Unlike existing methods, our approach is demonstrably able to learn the mapping between function spaces, and is invariant to different approximations and grids. It can do zero-shot super-resolution as demonstrated in Figure 2.1.
- By composing nonlinear activation functions and nonlocal integral operators, the learn operators can approximate complex nonlinear nonlocal solution operator raised in various PDEs.
- The standard integral operation requires quadratic time complexity in terms of the number of points, which is usually intractable. We study four efficient approximation schemes that have tractable complexity.
- Numerical results show that the Neural Operator has state-of-the-art accuracy comparing to previous data-driven finite-dimensional operators in the fixed-discretization setting. Meanwhile, it is also up to three order of magnitude faster than the conventional numerical solvers.

These concepts are illustrated in the context of a family of elliptic PDEs prototypical of a number of problems arising throughout the sciences and engineering.

2.2 Problem Setting

Our goal is to learn a mapping between two infinite dimensional spaces by using a finite collection of observations of input-output pairs from this mapping. Let $\mathcal{A}$ and $\mathcal{U}$ be separable Banach spaces and $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ a (typically) non-linear map. Suppose we have observations $\{a_j, u_j\}_{j=1}^N$ where $a_j \sim \mu$ is an i.i.d. sequence from the probability measure μ supported on $\mathcal{A}$ and $u_j = \mathcal{G}^\dagger(a_j)$ is possibly corrupted with noise. We aim to build an approximation of $\mathcal{G}^\dagger$ by constructing a parametric map

$$\mathcal{G} : \mathcal{A} \times \Theta \to \mathcal{U} \tag{2.2}$$

for some finite-dimensional parameter space Θ and then choosing $\theta^\dagger \in \Theta$ so that $\mathcal{G}(\cdot, \theta^\dagger) \approx \mathcal{G}^\dagger$.

This is a natural framework for learning in infinite-dimensions as one could define a cost functional $C : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ and seek a minimizer of the problem

$$\min_{\theta \in \Theta} \mathbb{E}_{a \sim \mu}[C(\mathcal{G}(a, \theta), \mathcal{G}^\dagger(a))]$$

which directly parallels the classical finite-dimensional setting (Vapnik, 1998). Showing the existence of minimizers, in the infinite-dimensional setting, remains a challenging open problem. We will approach this problem in the test-train setting in which empirical approximations to the cost are used. We conceptualize our methodology in the infinite-dimensional setting. This means that all finite-dimensional approximations can share a common set of network parameters which are defined in the (approximation-free) infinite-dimensional setting. To be concrete we will consider infinite-dimensional spaces which are Banach spaces of real-valued functions defined on a bounded open set in $\mathbb{R}^d$. We then consider mappings $\mathcal{G}^\dagger$ which take input functions to a PDE and map them to solutions of the PDE, both input and solutions being real-valued functions on $\mathbb{R}^d$.

A common instantiation of the preceding problem is the approximation of the second order elliptic PDE

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x), \quad && x \in D \\ u(x) &= 0, && x \in \partial D \end{aligned} \tag{2.3}$$

for some bounded, open set $D \subset \mathbb{R}^d$ and a fixed function $f \in L^2(D; \mathbb{R})$. This equation is prototypical of PDEs arising in numerous applications including hydrology (Bear and Corapcioglu, 2012) and elasticity (Antman, 2005). For a given $a \in \mathcal{A} = L^\infty(D; \mathbb{R}^+) \cap L^2(D; \mathbb{R}^+)$, equation (2.3) has a unique weak solution $u \in \mathcal{U} = H_0^1(D; \mathbb{R})$ (Evans, 2010) and therefore we can define the solution operator $\mathcal{G}^\dagger$ as the map $a \mapsto u$. Note that while the PDE (2.3) is linear, the solution operator $\mathcal{G}^\dagger$ is not.

Since our data a_j and u_j are , in general, functions, to work with them numerically, we assume access only to point-wise evaluations. To illustrate this, we will continue with the example of the preceding paragraph. To this end let $P_K = \{x_1, \ldots, x_K\} \subset D$ be a K-point discretization of the domain D and assume we have observations $a_j|_{P_K}, u_j|_{P_K} \in \mathbb{R}^K$, for a finite collection of input-output pairs indexed by j. In the next section, we propose a kernel inspired graph neural network architecture which, while trained on the discretized data, can produce an answer $u(x)$ for any $x \in D$ given a new input $a \sim \mu$. That is to say that our approach is independent of the

discretization P_K and therefore a true function space method; we verify this claim numerically by showing invariance of the error as $K \to \infty$. Such a property is highly desirable as it allows a transfer of solutions between different grid geometries and discretization sizes.

We note that, while the application of our methodology is based on having point-wise evaluations of the function, it is not limited by it. One may, for example, represent a function numerically as a finite set of truncated basis coefficients. Invariance of the representation would then be with respect to the size of this set. Our methodology can, in principle, be modified to accommodate this scenario through a suitably chosen architecture. We do not pursue this direction in the current work.

2.3 Graph Neural Operator

We propose the graph neural operator (graph kernel neural network) for the solution of the problem outlined in section 2.2. As a guiding principle for our architecture, we take the following example. Let $\mathcal{L}_a$ be a differential operator depending on a parameter $a \in \mathcal{A}$ and consider the PDE

$$\begin{aligned} (\mathcal{L}_a u)(x) &= f(x), \qquad && x \in D \\ u(x) &= 0, && x \in \partial D \end{aligned} \tag{2.4}$$

for a bounded, open set $D \subset \mathbb{R}^d$ and some fixed function f living in an appropriate function space determined by the structure of $\mathcal{L}_a$. The elliptic operator $\mathcal{L}_a \cdot = -\mathrm{div}(a\nabla\cdot)$ from equation (2.3) is an example. Under fairly general conditions on $\mathcal{L}_a$ (Evans, 2010), we may define the Green's function $G : D \times D \to \mathbb{R}$ as the unique solution to the problem

$$\mathcal{L}_a G(x, \cdot) = \delta_x$$

where δ_x is the delta measure on $\mathbb{R}^d$ centered at x. Note that G will depend on the parameter a thus we will henceforth denote it as G_a. The solution to (2.4) can then be represented as

$$u(x) = \int_D G_a(x, y) f(y)\, dy. \tag{2.5}$$

This is easily seen via the formal computation

$$\begin{aligned} (\mathcal{L}_a u)(x) &= \int_D (\mathcal{L}_a G(x, \cdot))(y) f(y)\, dy \\ &= \int_D \delta_x(y) f(y)\, dy \\ &= f(x) \end{aligned}$$

Generally the Green's function is continuous at points $x \neq y$, for example, when $\mathcal{L}_a$ is uniformly elliptic (Gilbarg and Trudinger, 2015), hence it is natural to model it via a neural network. Guided by the representation (3.1), we propose the following iterative architecture for $t = 0, \ldots, T-1$.

$$v_{t+1}(x) = \sigma\left(W v_t(x) + \int_D \kappa_\phi(x, y, a(x), a(y)) v_t(y)\, \nu_x(dy)\right) \tag{2.6}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is a fixed function applied element-wise, ν_x is a fixed Borel measure for each $x \in D$. The matrix $W \in \mathbb{R}^{n\times n}$, together with the parameters ϕ entering kernel $\kappa_\phi : \mathbb{R}^{2(d+1)} \to \mathbb{R}^{n\times n}$, are to be learned from data. We model κ_ϕ as a neural network mapping $\mathbb{R}^{2(d+1)}$ to $\mathbb{R}^{n\times n}$.

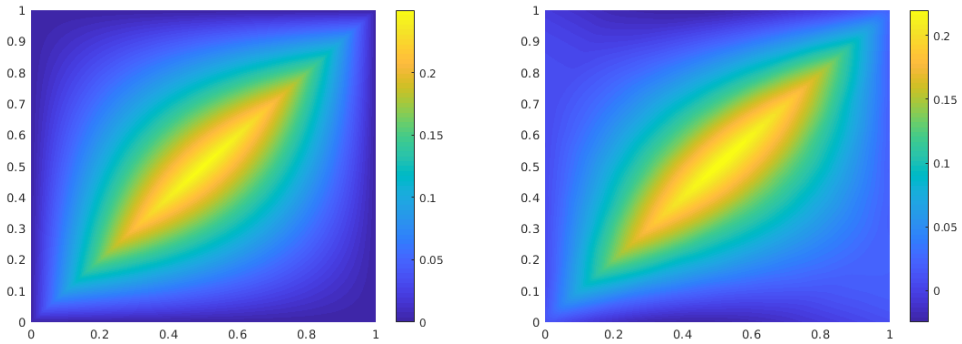
Discretization of the continuum picture may be viewed as replacing Borel measure ν_x by an empirical approximation based on the K grid points being used. In this setting we may view κ_ϕ as a $K \times K$ kernel block matrix, where each entry $\kappa_\phi(x, y)$ is itself a $n \times n$ matrix. Each block shares the same set of network parameters. This is the key to making a method that shares common parameters independent of the discretization used.

Finally, we observe that, although we have focussed on neural networks mapping a to u, generalizations are possible, such as mapping f to u, or having non-zero boundary data g on ∂D and mapping g to u. More generally one can consider the mapping from (a, f, g) into u and use similar ideas. Indeed to illustrate ideas we will consider the mapping from f to u below (which is linear and for which an analytic solution is known) before moving on to study the (nonlinear) mapping from a to u.

Example 1: Poisson equation. We consider a simplification of the foregoing in which we study the map from f to u. To this end we set $v_0(x) = f(x)$, $T = 1$, $n = 1$, $\sigma(x) = x$, $W = w = 0$, and $\nu_x(dy) = dy$ (the Lebesgue measure) in (2.6). We then obtain the representation (3.1) with the Green's function G_a parameterized by the neural network κ_ϕ with explicit dependence on $a(x), a(y)$. Now consider the setting where $D = [0, 1]$ and $a(x) \equiv 1$, so that (2.3) reduces to the 1-dimensional Poisson equation with explicitly computable Green's function. Indeed,

$$G(x, y) = \frac{1}{2}\left(x + y - |y - x|\right) - xy.$$

Note that although the map $f \mapsto u$ is, in function space, linear, the Green's function itself is not linear in either argument. Figure 2.2 shows κ_ϕ after training with



Proof of concept: graph kernel network on 1 dimensional Poisson equation; comparison of learned and truth kernel.

Figure 2.2: Kernel for one-dimensional Green's function

$N = 2048$ samples $f_j \sim \mu = \mathcal{N}(0, (-\Delta + I)^{-1})$ with periodic boundary conditions on the operator $-\Delta + I$. (samples from this measure can be easily implemented by means of a random Fourier series – Karhunen-Loeve – see (Lord, Powell, and Shardlow, 2014)).

Example 2: 2D Poisson equation. We further demonstrate the power of the graph kernel network by extending the Poisson equation studied in example 1 to the two dimensional (2D) case, where we approximate the map $f(x) \mapsto u(x)$ where $x \in D = [0, 1] \times [0, 1]$. We consider two approaches: (i) graph kernel network to approximate the 2D Green's function $G(x, y)$ and (ii) dense neural network with $f(x)$ as input and $u(x)$ as output such that the mapping $f(x) \mapsto u(x)$ is directly approximated.

The two neural networks are trained with the same training sets of different sizes ranging from 1 to 100 samples and tested with 1000 test samples. We observe that the Graph kernel network approximates the map with a minimum (≈ 5) number of training samples while still possessing a smaller test error comparing to that of a dense neural network trained with 100 samples. Therefore, the Graph kernel network potentially significantly reduces the number of required training samples to approximate the mapping. This property is especially important in practice as obtaining a huge number of training data for certain engineering/physics problems is always prohibitive.

The reason for the Graph kernel network to have such strong approximation power is because that it is able to learn the truth Green's function for the Poisson equation, as already demonstrated in the 1D case.

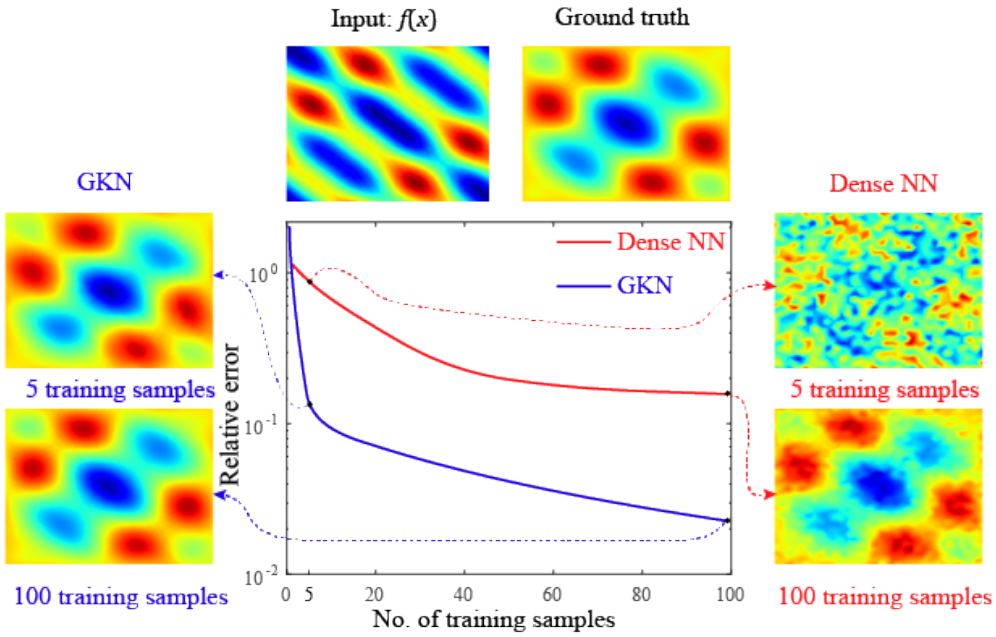


Figure 2.3: Comparison between the performance of graph kernel network and the dense neural network on the approximation of (2.3). Plots of relative l_2 test approximation errors versus the number of training samples for the Graph kernel network and dense neural networks when approximating the problem (2.3).

Algorithmic framework. The initialization $v_0(x)$ to our network (2.6) can be viewed as the initial guess we make for the solution $u(x)$ as well as any other dependence we want to make explicit. A natural choice is to start with the coefficient $a(x)$ itself as well as the position in physical space x. This $(d+1)$-dimensional vector field is then lifted to a n-dimensional vector field, an operation which we may view as the first layer of the overarching neural network. This is then used as an initialization to the kernel neural network, which is iterated T times. In the final layer, we project back to the scalar field of interest with another neural network layer.

Due to the smoothing effect of the inverse elliptic operator in (2.3) with respect to the input data a (and indeed f when we consider this as input), we augment the initialization $(x, a(x))$ with a Gaussian smoothed version of the coefficients $a_\epsilon(x)$, together with their gradient $\nabla a_\epsilon(x)$. Thus we initialize with a $2(d+1)$-dimensional vector field. Throughout this paper the Gaussian smoothing is performed with a centred isotropic Gaussian with variance 5. The Borel measure ν_x is chosen to be

the Lebesgue measure supported on a ball at x of radius r. Thus we have

$$v_0(x) = P(x, a(x), a_\epsilon(x), \nabla a_\epsilon(x)) + p \tag{2.7}$$

$$v_{t+1}(x) = \sigma\Big(W v_t(x) + \int_{B(x,r)} \kappa_\phi\big(x, y, a(x), a(y)\big) v_t(y)\, \mathrm{d}y\Big) \tag{2.8}$$

$$u(x) = Q v_T(x) + q \tag{2.9}$$

where $P \in \mathbb{R}^{n\times 2(d+1)}$, $p \in \mathbb{R}^n$, $v_t(x) \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{1\times n}$, $q \in \mathbb{R}$. The integration in (2.8) is approximated by a Monte Carlo sum via a message passing graph network with edge weights $(x, y, a(x), a(y))$. The choice of measure $\nu_x(\mathrm{d}y) = 1_{B(x,r)}\mathrm{d}y$ is two-fold: 1) it allows for more efficient computation and 2) it exploits the decay property of the Green's function. Note that if more information is known about the true kernel, it can be added into this measure. For example, if we know the true kernel has a Gaussian structure, we can define $\nu_x(\mathrm{d}y) = 1_{B(x,r)}\rho_x(y)\mathrm{d}y$ where $\rho_x(y)$ is a Gaussian density. Then κ_ϕ will need to learn a much less complicated function. We however do not pursue this direction in the current line of work.

Message passing graph networks. Message passing graph networks comprise a standard architecture employing edge features (Gilmer et al., 2017). If we properly construct the graph on the spatial domain D of the PDE, the kernel integration can be viewed as an aggregation of messages. Given node features $v_t(x) \in \mathbb{R}^n$, edge features $e(x, y) \in \mathbb{R}^{n_e}$, and a graph G, the message passing neural network with averaging aggregation is

$$v_{t+1}(x) = \sigma\Big(W v_t(x) + \frac{1}{|N(x)|}\sum_{y\in N(x)} \kappa_\phi\big(e(x, y)\big) v_t(y)\Big) \tag{2.10}$$

where $W \in \mathbb{R}^{n\times n}$, $N(x)$ is the neighborhood of x according to the graph, $\kappa_\phi\big(e(x, y)\big)$ is a neural network taking as input edge features and as output a matrix in $\mathbb{R}^{n\times n}$. Relating to (2.8), $e(x, y) = (x, y, a(x), a(y)) \in \mathbb{R}^{2(d+1)}$.

Graph construction. To use the message passing framework (3.4), we need to design a graph that connects the physical domain D of the PDE. The nodes are chosen to be the K discretized spatial locations. Here we work on a standard uniform mesh, but there are many other possibilities such as finite-element triangulations and random points at which data is acquired. The edge connectivity is then chosen according to the integration measure in (2.8), namely Lebesgue restricted to a ball. Each node $x \in \mathbb{R}^d$ is connected to nodes which lie within $B(x, r)$, defining the

neighborhood set $N(x)$. Then for each neighbor $y \in N(x)$, we assign the edge weight $e(x, y) = (x, y, a(x), a(y))$. Equation (3.4) can then be viewed as a Monte Carlo approximation of (2.8). This local structure allows for more efficient computation while remaining invariant to mesh-refinement. Indeed, since the radius parameter r is chosen in physical space, the size of the set $N(x)$ grows as the discretization size K grows. This is a key feature that makes our methodology mesh-independent.

Nyström approximation of the kernel. While the aforementioned graph structure severely reduces the computational overhead of integrating over the entire domain D (corresponding to a fully-connected graph), the number of edges still scales like $O(K^2)$. To overcome this, we employ a random Nyström-type approximation of the kernel. In particular, we uniformly sample $m \ll K$ nodes from the original graph, constructing a new random sub-graph. This process is repeated $l \in \mathbb{N}$ times, yielding l random sub-graphs each with m nodes. This can be thought of as a way of reducing the variance in the estimator. We use these sub-graphs when evaluating (3.4) during training, leading to the more favorable scaling $O(lm^2)$. Indeed, numerically we find that $l = 4$ and $m = 200$ is sufficient even when $K = 421^2 = 177,241$. In the evaluation phase, when we want the solution on a particular mesh geometry, we simply partition the mesh into sub-graphs each with m nodes and evaluate each separately.

We will now highlight the quality of this kernel approximation in a RHKS setting. A real Reproducing Kernel Hilbert Space (RKHS) $(\mathcal{H}, \langle\cdot,\cdot\rangle, \|\cdot\|)$ is a Hilbert space of functions $f : D \to \mathbb{R}$ where point-wise evaluation is a continuous linear functional, i.e. $|f(x)| \leq C\|f\|$ for some constant $C \geq 0$, independent of x. For every RHKS, there exists a unique, symmetric, positive definite kernel $\kappa : D \times D \to \mathbb{R}$, which gives the representation $f(x) = \langle f, \kappa(\cdot, x)\rangle$. Let $T : \mathcal{H} \to \mathcal{H}$ be a linear operator on $\mathcal{H}$ acting via the kernel

$$Tf = \int_{B(\cdot,r)} \kappa(\cdot, y) f(y) \nu(dy).$$

Let $T_m : \mathcal{H} \to \mathcal{H}$ be its m-point empirical approximation

$$T_m = \int_{B(\cdot,r)} \kappa(\cdot, y) f(y) \nu_m(dy),$$
$$\nu_m(dy) = \frac{1}{m} \sum_{k=1}^{m} \delta_{y_k}(dy),$$

so that

$$T_m f = \frac{1}{m} \sum_{k=1}^{m} \kappa(\cdot, y_k) f(y_k).$$

The error of this approximation achieves the Monte Carlo rate $O(m^{-1/2})$:

Proposition 2.3.1 *Suppose $\mathbb{E}_{y\sim\nu}[\kappa(\cdot, y)^4] < \infty$ then there exists a constant $C \geq 0$ such that*

$$\mathbb{E}\|T - T_m\|_{HS} \leq \frac{C}{\sqrt{m}}$$

where $\|\cdot\|_{HS}$ denotes the Hilbert-Schmidt norm on operators acting on $\mathcal{H}$.

With stricter assumptions similar results can also be proven with high probability (Rosasco, Belkin, and Vito, 2010).

Lemma 2.3.1 *(Rosasco, Belkin, and Vito, 2010) T and T_K are Hilbert-Schmidt. Furthermore, with probability greater than $1 - 2e^{\tau}$*

$$\|T - T_K\|_{HS} \leq \frac{2\sqrt{2}k\sqrt{\tau}}{\sqrt{K}}$$

where $k = \sup_{x\in D} \kappa(x, x)$.

We note that, in our algorithm, $\kappa : D \times D \to \mathbb{R}^{n\times n}$ whereas the preceding results are proven only in the setting $n = 1$; nonetheless they provide useful intuition regarding the approximations used in our methodology.

2.4 Experiments

In this section we illustrate the claimed properties of our methodology, and compare it to existing approaches in the literature. All experimental results concern the mapping $a \mapsto u$ defined by (2.3) with $D = [0, 1]^2$. Coefficients are generated according to $a \sim \mu$ where $\mu = \psi_{\#}\mathcal{N}(0, (-\Delta + 9I)^{-2})$ with a Neumann boundry condition on the operator $-\Delta + 9I$. The mapping $\psi : \mathbb{R} \to \mathbb{R}$ takes the value 12 on the positive part of the real line and 3 on the negative. Hence the coefficients are piecewise constant with random geometry and a fixed contrast of 4. Such constructions are prototypical of physical properties such as permeability in subsurface flows and material microstructures in elasticity. Solutions u are obtained by using a second-order finite difference scheme on a 241×241 grid. Different resolutions are downsampled from this dataset.

Resolutions	$s' = 16$	$s' = 31$	$s' = 61$
$s = 16$	0.0525	0.0591	0.0585
$s = 31$	0.0787	0.0538	0.0588

$r = 0.10$, $N = 100$, relative l_2 test error

Table 2.1: Comparing resolutions on full grids

To be concrete we set the dimension of representation n (i.e. the width of graph network) to be 64, the number of iterations T to be 6, σ to be the ReLU function, and the inner kernel network κ to be a 3−layer feed-forward network with widths $(6, 512, 1024, n^2)$ and ReLU activation. We use the Adam optimizer with a learning rate $1e-4$ and train for 200 epochs with respect to the absolute mean squared error on the normalized data unless otherwise stated. These chosen hyperparameters are not optimized and could be adapted to improve performance. We employ the message passing network from the standard Pytorch graph network library Torch-geometric (Fey and Lenssen, 2019). All the test errors are relative $L^2(D)$ errors on the original data. The **code** and **data** can be found at `https://github.com/wumming/graph-pde`.

Supervised Setting

First we consider the supervised scenario that we are given N training pairs $\{a_j, u_j\}_1^N$, where each a_j and u_j are provided on a $s \times s$ grid ($K = s^2$).

Generalization of resolutions on full grids. To examine the generalization property, we train the graph kernel network on resolution $s \times s$ and test on another resolution $s' \times s'$. We fix the radius to be $r = 0.10$, train on $N = 100$ equation pairs and test on 40 equation pairs.

As shown in Table 2.1, for each row, the test errors at different resolutions remain on the same scale, illustrating the desired design feature that graph kernel networks can train on one resolution and generalize to another resolution. The test errors on the diagonal ($s = s' = 16$ and $s = s' = 31$) are the smallest, which means the network has the best performance when the training grid and the test grid are the same. Interestingly, for the second row, when training on $s = 31$, it is easier to general to $s' = 61$ than to $s' = 16$. This is because when generalizing to a larger grid, the support of the kernel becomes large which does not hurt the performance. But when generalizing to a smaller grid, part of the support of the kernel is lost, which causes the kernel to be inaccurate.

Training Size	Training Error	Test Error
$N = 10$	0.0089	0.0931
$N = 100$	0.0183	0.0478
$N = 1000$	0.0255	0.0345

2000, 500, 100 epochs, respectively.

Table 2.2: Comparing number of training pairs

Expressiveness and overfitting. We compare the training error and test error with a different number of training pairs N to see if the kernel network can learn the kernel structure even with a small amount of data. We study the expressiveness of the kernel network, examining how it overfits. We fix $r = 0.10$ on the $s = s' = 31$ grid and train with $N = 10, 100, 1000$ whilst employing 2000, 500, 100 epochs respectively.

We see from Table 2.2 that the kernel network already achieves a reasonable result when $N = 10$, and the accuracy is competitive when $N = 100$. In all three cases, the test error is larger than the training error suggesting that the kernel network has enough expressiveness to overfit the training set. This overfitting is not severe as the training error will not be pushed to zero even for $N = 10$, after 2000 epochs.

Semi-Supervised Setting

In the semi-supervised setting, we are only given m nodes sampled from a $s \times s$ grid for each training pair, and want to evaluate on m' nodes sampled from an $s' \times s'$ grid for each test pair. To be concrete, we set the number of sampled nodes $m = m' = 200$. For each training pair, we sample twice $l = 2$; for each test pair, we sample once $l' = 1$. We train on $N = 100$ pairs and test on $N' = 100$ pairs. The radius for both training and testing is set to $r = r' = 0.25$.

Generalization of resolutions on sampled grids. Similar to the first experiments, we train the graph kernel network with nodes sampled from the $s \times s$ resolution and test on nodes sampled from the $s' \times s'$ resolution. As shown in Table 2.3, for each row, the test errors on different resolutions are about the same, which means the graph kernel network can also generalize in the semi-supervised setting. Comparing the rows, large training resolutions s tend to have a smaller error. When sampled from a finer grid, there are more edges because the support of the kernel is larger on the finer grid. Still, the performance is best when $s = s'$.

Resolutions	$s' = 61$	$s' = 121$	$s' = 241$
$s = 16$	0.0717	0.0768	0.0724
$s = 31$	0.0726	0.0710	0.0722
$s = 61$	0.0687	0.0728	0.0723
$s = 121$	0.0687	0.0664	0.0685
$s = 241$	0.0649	0.0658	0.0649

$N = 100$, $m = m' = 200$, $r = r' = 0.25$, $l = 2$

Table 2.3: Generalization of resolutions on sampled Grids

The number of examples vs the times of sampling. Increasing the number of times we sample, l, reduces the error from the Nyström approximation. By comparing different l we determine which value will be sufficient. When we sample l times for each equation, we get a total of Nl sampled training pairs, Table 2.4.

	$l = 1$	$l = 2$	$l = 4$	$l = 8$
$N = 10$	0.1259	0.1069	0.0967	0.1026
$N = 100$	0.0786	0.0687	0.0690	0.0621
$N = 1000$	0.0604	0.0579	0.0540	0.0483

$s = 241$, $m = m' = 200$, $r = r' = 0.25$

Table 2.4: Number of training pairs and sampling

Table 2.4 indicates that the larger l the better, but $l = 2$ already gives a reasonable performance. Moreover, the same order of sampled training pairs, $(N = 100, l = 8)$, and $(N = 1000, l = 1)$, result in a similar performance. It implies that in a low training data regime, increasing l improves the performance.

Different number of nodes in training and testing. To further examine the Nyström approximation, we compare different numbers of node samples m, m' for both training and testing.

	$m' = 100$	$m' = 200$	$m' = 400$	$m' = 800$
$m = 100$	0.0871	0.0716	0.0662	0.0609
$m = 200$	0.0972	0.0734	0.0606	0.0562
$m = 400$	0.0991	0.0699	0.0560	0.0506
$m = 800$	0.1084	0.0751	0.0573	0.0478

$s = 121, r = r' = 0.15, l = 5$

Table 2.5: Number of nodes in the training and testing

As can be seen from Table 2.5, in general the larger m and m' the better. For each row, fixing m, the larger m' the better. But for each column, when fixing m', increasing m may not lead to better performance. This is again due to the fact that when learning on a larger grid, the kernel network learns a kernel with larger support. When evaluating on a smaller grid, the learned kernel will be truncated to have small support, leading to an increased error. In general, $m = m'$ will be the best choice.

The radius and the number of nodes. The computation and storage of graph networks directly scale with the number of edges. In this experiment, we want to study the trade-off between the number of nodes m and the radius r when fixing the number of edges.

	$m = 100$	$m = 200$	$m = 400$
$r = 0.05$	0.110(176)	0.109(666)	0.099(3354)
$r = 0.15$	0.086(512)	0.070(2770)	0.053(14086)
$r = 0.40$	0.064(1596)	0.051(9728)	0.040(55919)
$r = 1.00$	0.059(9756)	0.048(38690)	–

Error (Edges), $s = 121$, $l = 5$, $m' = m$

Table 2.6: The radius and the number of nodes

Table 2.6 shows the test error with the number of edges for different r and m. In general, the more edges, the better. For a fixed number of edges, the performance depends more on the radius r than on the number of nodes m. In other words, the error of truncating the kernel locally is larger than the error from the Nyström approximation. It would be better to use larger r with smaller m.

Inner Kernel Network κ. To find the best network structure of κ, we compare different combinations of width and depth. We consider three cases of κ: 1. a 2−layer feed-forward network with widths $(6, width, n^2)$, 2. a 3−layer feed-forward network with widths $(6, width/2, width, n^2)$, and 3. a 5−layer feed-forward network with widths $(6, width/4, width/2, width, width, n^2)$, all with ReLU activation and learning rate $1e-4$.

As shown in Table 2.7, have wider and deeper network increase the expensiveness of the kernel network. The diagonal combinations $width = 256, depth = 2$, $width = 1024, depth = 3$, and $width = 4096, depth = 5$ have better test error. Notice the wide but shallow network $width = 4096, depth = 2$ has very bad

	depth = 2	depth = 3	depth = 5
width = 64	0.0685	0.0695	0.0770
width = 128	0.0630	0.0633	0.0702
width = 256	0.0617	0.0610	0.0688
width = 1024	0.0641	0.0591	0.0608
width = 4096	0.2934	0.0690	0.0638

$s = 241$, $m = 200$, $r = 0.25$

Table 2.7: Comparing the width and depth for the inner kernel network κ

performance. Both its training and testing error once decreased to 0.09 around 10th epochs, but then blow off. The $1e-4$ learning rate is probably too high for this combination. In general, depth of 3 with with of 256 is a good combination of Dracy Equation dataset.

Comparison with Different Benchmarks

In the following section, we compare the Graph Kernel Network with different benchmarks on a larger dataset of $N = 1024$ training pairs computed on a 421×421 grid. The network is trained and evaluated on the same full grid. The results are presented in Table 2.8.

- **NN** is a simple point-wise feedforward neural network. It is mesh-free, but performs badly due to lack of neighbor information.
- **FCN** is the state of the art neural network method based on Fully Convolution Network (Zhu and Zabaras, 2018). It has a dominating performance for small grids $s = 61$. But fully convolution networks are mesh-dependent and therefore their error grows when moving to a larger grid.
- **PCA+NN** is an instantiation of the methodology proposed in (Bhattacharya, Kovachki, and Stuart, 2020): using PCA as an autoencoder on both the input and output data and interpolating the latent spaces with a neural network. The method provably obtains mesh-independent error and can learn purely from data, however the solution can only be evaluated on the same mesh as the training data.
- **RBM** is the classical Reduced Basis Method (using a PCA basis), which is widely used in applications and provably obtains mesh-independent error (R. A. DeVore, 2014). It has the best performance but the solutions can only

be evaluated on the same mesh as the training data and one needs knowledge of the PDE to employ it.

- **GKN** stands for our graph kernel network with $r = 0.25$ and $m = 300$. It enjoys competitive performance against all other methods while being able to generalize to different mesh geometries.

Networks	$s = 85$	$s = 141$	$s = 211$	$s = 421$
NN	0.1716	0.1716	0.1716	0.1716
FCN	0.0253	0.0493	0.0727	0.1097
PCA+NN	0.0299	0.0298	0.0298	0.0299
RBM	0.0244	0.0251	0.0255	0.0259
GKN	0.0346	0.0332	0.0342	0.0369

Table 2.8: Error of different methods

2.5 Discussion and Conclusion

We have introduced the concept of Neural Operator and instantiated it through *graph kernel networks* designed to approximate mappings between function spaces. They are constructed to be mesh-free and our numerical experiments demonstrate that they have the desired property of being able to train and generalize on different meshes. This is because the networks learn the mapping between infinite-dimensional function spaces, which can then be shared with approximations at various levels of discretization. A further advantage is that data may be incorporated on unstructured grids, using the Nyström approximation. We demonstrate that our method can achieve competitive performance with other mesh-free approaches developed in the numerical analysis community, and beats state-of-the-art neural network approaches on large grids, which are mesh-dependent. The methods developed in the numerical analysis community are less flexible than the approach we introduce here, relying heavily on the variational structure of divergence form elliptic PDEs. Our new mesh-free method has many applications. It has the potential to be a faster solver that learns from only sparse observations in physical space. It is the only method that can work in the semi-supervised scenario when we only have measurements on some parts of the grid. It is also the only method that can transfer between different geometries. For example, when computing the flow dynamic of many different airfoils, we can construct different graphs and train together. When learning from irregular grids and querying new locations, our method does not require any interpolation, avoid subsequently interpolation error.

Disadvantage. Graph kernel network's runtime and storage scale with the number of edges $E = O(K^2)$. While other mesh-dependent methods such as PCA+NN and RBM require only $O(K)$. This is somewhat inevitable, because to learn the continuous function or the kernel, we need to capture pairwise information between every two nodes, which is $O(K^2)$; when the discretization is fixed, one just needs to capture the point-wise information, which is $O(K)$. Therefore training and evaluating the whole grid is costly when the grid is large. On the other hand, subsampling to ameliorate cost loses information in the data, and causes errors which make our method less competitive than PCA+NN and RBM.

Future works. To deal with the above problem, we propose that ideas such as multi-grid and fast multipole methods (Gholami et al., 2016) may be combined with our approach to reduce complexity. In particular, a multi-grid approach will construct multi-graphs corresponding to different resolutions so that, within each graph, nodes only connect to their nearest neighbors. The number of edges then scale as $O(K)$ instead of $O(K^2)$. The error terms from Nyström approximation and local truncation can be avoided. Another direction is to extend the framework for time-dependent PDEs. Since the graph kernel network is itself an iterative solver with the time step t, it is natural to frame it as an RNN that each time step corresponds to a time step of the PDEs.

References

Adler, Jonas and Ozan Oktem (Nov. 2017). "Solving ill-posed inverse problems using iterative deep neural networks". In: *Inverse Problems*. DOI: 10.1088/1361-6420/aa9581. URL: https://doi.org/10.1088%2F1361-6420%2Faa9581.

Alet, Ferran et al. (2019). "Graph Element Networks: adaptive, structured computation and memory". In: *36th International Conference on Machine Learning*. PMLR. URL: http://proceedings.mlr.press/v97/alet19a.html.

Antman, Stuart S (2005). *Problems In Nonlinear Elasticity*. Springer.

Bar, Leah and Nir Sochen (2019). "Unsupervised deep learning algorithm for PDE-based forward and inverse problems". In: *arXiv preprint arXiv:1904.05417*.

Battaglia, Peter W et al. (2018). "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261*.

Bear, Jacob and M Yavuz Corapcioglu (2012). *Fundamentals of transport phenomena in porous media*. Springer Science & Business Media.

Belongie, Serge et al. (2002). "Spectral partitioning with indefinite kernels using the Nyström extension". In: *European conference on computer vision*. Springer.

Bhatnagar, Saakaar et al. (2019). "Prediction of aerodynamic flow fields using convolutional neural networks". In: *Computational Mechanics*, pp. 1–21.

Bhattacharya, Kaushik, Nikola B. Kovachki, and Andrew M. Stuart (2020). "Model Reduction and Neural Networks for Parametric PDE(s)". In: *arXiv preprint arXiv:2005.03180*.

Chen, Chi et al. (2019). "Graph networks as a universal machine learning framework for molecules and crystals". In: *Chemistry of Materials* 31.9, pp. 3564–3572.

Cohen, Albert and Ronald DeVore (2015). "Approximation of high-dimensional parametric PDEs". In: *Acta Numerica*. DOI: 10.1017/S0962492915000033.

Damianou, Andreas and Neil Lawrence (2013). "Deep gaussian processes". In: *Artificial Intelligence and Statistics*, pp. 207–215.

DeVore, Ronald A. (2014). "Chapter 3: The Theoretical Foundation of Reduced Basis Methods". In: *Model Reduction and Approximation*. DOI: 10.1137/1.9781611974829.ch3. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611974829.ch3. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611974829.ch3.

E, Weinan and Bing Yu (Mar. 2018). "The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems". English (US). In: *Communications in Mathematics and Statistics*. ISSN: 2194-6701. DOI: 10.1007/s40304-018-0127-z.

Evans, Lawrence C (2010). *Partial Differential Equations*. Vol. 19. American Mathematical Soc.

Fey, Matthias and Jan E. Lenssen (2019). "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning*.

Garriga-Alonso, Adrià, Carl Edward Rasmussen, and Laurence Aitchison (Aug. 2018). "Deep Convolutional Networks as shallow Gaussian Processes". In: *arXiv e-prints*, arXiv:1808.05587, arXiv:1808.05587. arXiv: 1808.05587 [stat.ML].

Gholami, Amir et al. (2016). "FFT, FMM, or multigrid? A comparative study of state-of-the-art Poisson solvers for uniform and nonuniform grids in the unit cube". In: *SIAM Journal on Scientific Computing*.

Gilbarg, David and Neil S Trudinger (2015). *Elliptic partial differential equations of second order*. springer.

Gilmer, Justin et al. (2017). "Neural message passing for quantum chemistry". In: *Proceedings of the 34th International Conference on Machine Learning*.

Globerson, Amir and Roi Livni (2016). "Learning Infinite-Layer Networks: Beyond the Kernel Trick". In: *CoRR* abs/1606.05316. arXiv: 1606.05316. URL: http://arxiv.org/abs/1606.05316.

Guo, Xiaoxiao, Wei Li, and Francesco Iorio (2016). "Convolutional neural networks for steady flow approximation". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Gurtin, Morton E (1982). *An introduction to continuum mechanics*. Academic press.

Guss, William H. (Dec. 2016). "Deep Function Machines: Generalized Neural Networks for Topological Layer Expression". In: *arXiv e-prints*, arXiv:1612.04799, arXiv:1612.04799. arXiv: 1612.04799 [stat.ML].

Hamilton, Will, Zhitao Ying, and Jure Leskovec (2017). "Inductive representation learning on large graphs". In: *Advances in neural information processing systems*, pp. 1024–1034.

Herrmann, L, Ch Schwab, and J Zech (2020). "Deep ReLU Neural Network Expression Rates for Data-to-QoI Maps in Bayesian PDE Inversion". In.

Johnson, Claes (2012). *Numerical solution of partial differential equations by the finite element method*. Courier Corporation.

Kipf, Thomas N and Max Welling (2016). "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907*.

Lord, Gabriel J, Catherine E Powell, and Tony Shardlow (2014). *An introduction to computational stochastic PDEs*. Vol. 50. Cambridge University Press.

Matthews, Alexander G. de G. et al. (Apr. 2018). "Gaussian Process Behaviour in Wide Deep Neural Networks". In.

Murphy, Ryan L et al. (2018). "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs". In: *arXiv preprint arXiv:1811.01900*.

Neal, Radford M. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag. ISBN: 0387947248.

Nyström, Evert J (1930). "Über die praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben". In: *Acta Mathematica*.

Raissi, Maziar, Paris Perdikaris, and George E Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378, pp. 686–707.

Rosasco, Lorenzo, Mikhail Belkin, and Ernesto De Vito (Mar. 2010). "On Learning with Integral Operators". In: *J. Mach. Learn. Res.* 11, pp. 905–934. ISSN: 1532-4435.

Roux, Nicolas Le and Yoshua Bengio (2007). "Continuous Neural Networks". In: *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*. Ed. by Marina Meila and Xiaotong Shen.

Ummenhofer, Benjamin et al. (2020). "Lagrangian fluid simulation with continuous convolutions". In: *International Conference on Learning Representations*.

Vapnik, Vladimir N. (1998). *Statistical Learning Theory*. Wiley-Interscience.

Veličković, Petar et al. (2017). "Graph attention networks". In.

Williams, Christopher K. I. (1996). "Computing with Infinite Networks". In: *Proceedings of the 9th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press.

Zhu, Yinhao and Nicholas Zabaras (2018). "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification". In: *Journal of Computational Physics*. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.04.018. URL: http://www.sciencedirect.com/science/article/pii/S0021999118302341.

Chapter 3

MODEL: MULTIPOLE NEURAL OPERATOR

One of the main challenges in using deep learning-based methods for simulating physical systems and solving partial differential equations (PDEs) is formulating physics-based data in the desired structure for neural networks. Graph neural networks (GNNs) have gained popularity in this area since graphs offer a natural way of modeling particle interactions and provide a clear way of discretizing the continuum models. However, the graphs constructed for approximating such tasks usually ignore long-range interactions due to unfavorable scaling of the computational complexity with respect to the number of nodes. The errors due to these approximations scale with the discretization of the system, thereby not allowing for generalization under mesh-refinement. Inspired by the classical multipole methods, we propose a novel multi-level graph neural network framework that captures interaction at all ranges with only linear complexity. Our multi-level formulation is equivalent to recursively adding inducing points to the kernel matrix, unifying GNNs with multi-resolution matrix factorization of the kernel. Experiments confirm our multi-graph network learns discretization-invariant solution operators to PDEs and can be evaluated in linear time.

3.1 Introduction

A wide class of important scientific applications involve numerical approximation of parametric PDEs. There has been immense research efforts in formulating and solving the governing PDEs for a variety of physical and biological phenomena ranging from the quantum to the cosmic scale. While this endeavor has been successful in producing solutions to real-life problems, major challenges remain. Solving complex PDE systems such as those arising in climate modeling, turbulent flow of fluids, and plastic deformation of solid materials requires considerable time, computational resources, and domain expertise. Producing accurate, efficient, and automated data-driven approximation schemes has the potential to significantly accelerate the rate of innovation in these fields. Machine learning based methods enable this since they are much faster to evaluate and require only observational data to train, in stark contrast to traditional Galerkin methods (Zienkiewicz et al., 1977) and classical reduced order models (Rozza, Huynh, and Patera, 2007).

While deep learning approaches such as convolutional neural networks can be fast and powerful, they are usually restricted to a specific format or discretization. On the other hand, many problems can be naturally formulated on graphs. An emerging class of neural network architectures designed to operate on graph-structured data, Graph neural networks (GNNs), have gained popularity in this area. GNNs have seen numerous applications on tasks in imaging, natural language modeling, and the simulation of physical systems (Wu et al., 2020). In the latter case, graphs are typically used to model particles systems (the nodes) and the their interactions (the edges). Recently, GNNs have been directly used to learn solutions to PDEs by constructing graphs on the physical domain (Alet et al., 2019), and it is was further shown that GNNs can learn mesh-invariant solution operators (Z. Li et al., 2020). Since GNNs offer great flexibility in accurately representing solutions on any unstructured mesh, finding efficient algorithms is an important open problem.

The computational complexity of GNNs depends on the sparsity structure of the underlying graph, scaling with the number of edges which may grow quadratically with the number of nodes in fully connected regions (Wu et al., 2020). Therefore, to make computations feasible, GNNs make approximations using nearest neighbor connection graphs which ignore long-range correlations. Such approximations are not suitable in the context of approximating solution operators of parametric PDEs since they will not generalize under refinement of the discretization, as we demonstrate in Section 8.5. However, using fully connected graphs quickly becomes computationally infeasible. Indeed evaluation of the kernel matrices outlined in Section 3.3 is only possible for coarse discretizations due to both memory and computational constraints. Throughout this work, we aim to develop approximation techniques that help alleviate this issue.

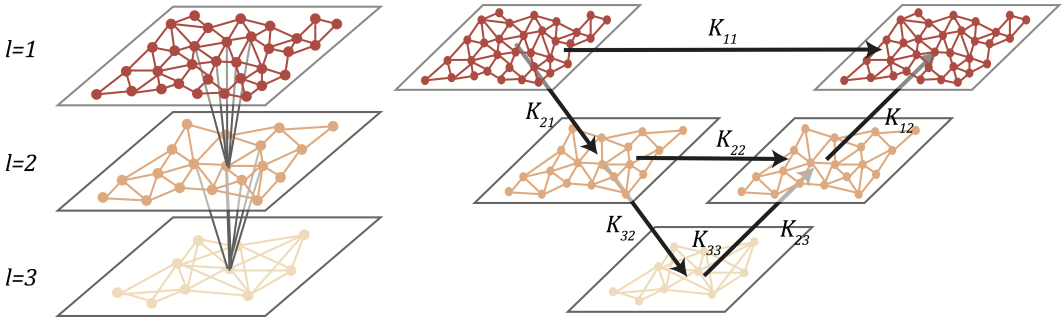
To efficiently capture long-range interaction, multi-scale methods such as the classical fast multipole methods (FMM) have been developed (Greengard and Rokhlin, 1997). Based on the insight that long-range interaction are smooth, FMM decomposes the kernel matrix into different ranges and hierarchically imposes low-rank structures to the long-range components (hierarchical matrices)(Börm, Grasedyck, and Hackbusch, 2003). This decomposition can be viewed as a specific form of the multi-resolution matrix factorization of the kernel (Kondor, Teneva, and Garg, 2014; Börm, Grasedyck, and Hackbusch, 2003). However, the classical FMM requires nested grids as well as the explicit form of the PDEs. We generalize this idea to arbitrary graphs in the data-driven setting, so that the corresponding graph neural

networks can learn discretization-invariant solution operators.

Main contributions. Inspired by the fast multi-pole method (FMM), we propose a novel hierarchical, and multi-scale graph structure which, when deployed with GNNs, captures global properties of the PDE solution operator with a linear time-complexity (Greengard and Rokhlin, 1997; L. Ying, Biros, and Zorin, 2004). As shown in Figure 3.1, starting with a nearest neighbor graph, instead of directly adding edges to connect every pair of nodes, we add inducing points which help facilitate long-range communication. The inducing points may be thought of as forming a new subgraph which models long-range correlations. By adding a small amount of inducing nodes to the original graph, we make computation more efficient. Repeating this process yields a hierarchy of new subgraphs, modeling correlations at different length scales.

We show that message passing through the inducing points is equivalent to imposing a low-rank structure on the corresponding kernel matrix, and recursively adding inducing points leads to multi-resolution matrix factorization of the kernel (Kondor, Teneva, and Garg, 2014; Börm, Grasedyck, and Hackbusch, 2003). We propose the graph V-cycle algorithm (Figure 3.1) inspired by FMM, so that message passing through the V-cycle directly computes the multi-resolution matrix factorization. We show that the computational complexity of our construction is linear in the number of nodes, achieving the desired efficiency, and we demonstrate the linear complexity and competitive performance through experiments on Darcy flow (Tek et al., 1957), a linear second-order elliptic equation, and Burgers' equation(Su and Gardner, 1969), which considered a stepping stone to Naiver-Stokes, is nonlinear, long-range correlated and more challenging. Our primary contributions are listed below.

- We develop the multipole graph kernel neural network (MGKN) that can capture long-range correlations in graph-based data with a linear time complexity in the nodes of the graph.
- We unify GNNs with multi-resolution matrix factorization through the V-cycle algorithm.
- We verify, analytically and numerically, the linear time complexity of our proposed methodology.



Left: the multi-level graph. **Right**: one V-cycle iteration for the multipole graph kernel network.

Figure 3.1: V-Cycle Algorithm on Graph

- We demonstrate numerically our method's ability to capture global information by learning mesh-invariant solution operators to the Darcy flow and Burgers' equations.

3.2 Related Works

Deep learning approaches for PDEs: There have been two primary approaches in the application of deep learning for the solution of PDEs. The first parametrizes the solution operator as a deep convolutional neural network (CNN) between finite-dimensional Euclidean spaces $\mathcal{G}_\theta : \mathbb{R}^n \to \mathbb{R}^n$ (Guo, W. Li, and Iorio, 2016; Zhu and Zabaras, 2018; Adler and Oktem, 2017; Bhatnagar et al., 2019; Ummenhofer et al., 2020). As demonstrated in Figure 3.4, such approaches are tied to a discritezation and cannot generalize. The second approach directly parameterizes the solution u as a neural network $\mathcal{G}_\theta : D \to \mathbb{R}$ (E and Yu, 2018; Raissi, Perdikaris, and George E Karniadakis, 2019; Bar and Sochen, 2019; Smith, Azizzadenesheli, and Ross, 2020; Jiang et al., 2020). This approach is close to classical Galerkin methods and therefore suffers from the same issues w.r.t. the parametric dependence in the PDE. For any new parameter, an optimization problem must be solved which requires backpropagating through the differential operator $\mathcal{L}_a$ many times, making it too slow for many practical application. Only very few recent works have attempted to capture the infinite-dimensional solution operator of the PDE (Alet et al., 2019; Lu, Jin, and George Em Karniadakis, 2019; Z. Li et al., 2020; Bhattacharya, Kovachki, and Andrew M. Stuart, 2020; Nelsen and Andrew M Stuart, 2020). The current work advances this direction.

GNN and non-sparse graphs: A multitude of techniques such as graph convolution, edge convolution, attention, and graph pooling, have been developed for

improving GNNs (Kipf and Welling, 2016; Hamilton, Z. Ying, and Leskovec, 2017; Gilmer et al., 2017; Veličković et al., 2017; Murphy et al., 2018). Most, however, have been designed for sparse graphs and become computationally infeasible as the number of edges grow. The work (Alfke and Stoll, 2019) proposes using a low-rank decomposition to address this issue. Works on multi-resolution graphs with a U-net like structure have also recently began to emerge and seen success in imaging, classification, and semi-supervised learning (Ronneberger, Fischer, and Brox, 2015; Abu-El-Haija, Kapoor, et al., 2018; Abu-El-Haija, Perozzi, et al., 2019; Gao and Ji, 2019; M. Li et al., 2020). Our work ties together many of these ideas and provides a principled way of designing multi-scale GNNs(Gao and Ji, 2019; Abu-El-Haija, Kapoor, et al., 2018; Abu-El-Haija, Perozzi, et al., 2019; M. Li et al., 2020). All of these works focus on build multi-scale structure on a given graph. Our method, on the other hand, studies how to construct randomized graphs on the spatial domain for physics and applied math problems. We carefully craft the multi-level graph that corresponds to multi-resolution decomposition of the kernel matrix.

Multipole and multi-resolution methods: The works (Fan, Lin, et al., 2019; Fan, Feliu-Faba, et al., 2019) propose a similar multi-pole expansion for solving parametric PDEs on structured grids. Our work generalizes on this idea by allowing for arbitrary discretizations through the use of GNNs. Multi-resolution matrix factorizations have been proposed in (Kondor, Teneva, and Garg, 2014; Ithapu et al., 2017). We employ such ideas to build our approximation architecture.

3.3 Preliminaries

Graph Kernel Network (GKN)

Suppose that $\mathcal{L}_a$ in (2.4) is uniformly elliptic then the Green's representation formula implies

$$u(x) = \int_D G_a(x, y)[f(y) + (\Gamma_a u)(y)]\, dy. \tag{3.1}$$

where G_a is a Newtonian potential and Γ_a is an operator defined by appropriate sums and compositions of the modified trace and co-normal derivative operators (Sauter and Schwab, 2010). We have turned the PDE (2.4) into the integral equation (3.1) which lends itself to an iterative approximation architecture.

Kernel operator. Since G_a is continuous for all points $x \neq y$, it is sensible to model the action of the integral operator in (3.1) by a neural network κ_ϕ with parameters ϕ. To that end, define the operator $\mathcal{K}_a : \mathcal{U} \to \mathcal{U}$ as the action of the

kernel κ_ϕ on u:

$$(\mathcal{K}_a u)(x) = \int_D \kappa_\phi(a(x), a(y), x, y)u(y)\, dy \tag{3.2}$$

where the kernel neural network κ_ϕ takes as inputs spatial locations x, y as well as the values of the parameter $a(x), a(y)$. Since Γ_a is itself an operator, its action cannot be fully accounted for by the kernel κ_ϕ, we therefore add local parameters $W = wI$ to $\mathcal{K}_a$ and apply a non-linear activation σ, defining the iterative architecture $u^{(t)} = \sigma((W + \mathcal{K}_a)u^{(t-1)})$ for $t = 1, \dots T$ with $u^{(0)} = a$. Since Γ_a is local w.r.t u, we only need local parameters to capture its effect, while we expect its non-locality w.r.t. a to manifest via the initial condition (Sauter and Schwab, 2010). To increase expressivity, we lift $u(x) \in \mathbb{R}$ to a higher dimensional representation $v(x) \in \mathbb{R}^{d_v}$ by a point-wise linear transformation $v_0 = Pu_0$, and update the representation

$$v^{(t)} = \sigma\big((W + \mathcal{K}_a)v^{(t-1)}\big), \qquad t = 1, \dots, T \tag{3.3}$$

projecting it back $u^{(T)} = Qv^{(T)}$ at the last step. Hence the kernel is a mapping $\kappa_\phi : \mathbb{R}^{2(d+1)} \to \mathbb{R}^{d_v \times d_v}$ and $W \in \mathbb{R}^{d_v \times d_v}$. Note that, since our goal is to approximate the mapping $a \mapsto u$ with f in (2.4) fixed, we do not need explicit dependence on f in our architecture as it will remain constant for any new $a \in \mathcal{A}$.

For a specific discretization D_j, $a_j|_{D_j}, u_j|_{D_j} \in \mathbb{R}^n$ are n-dimensional vectors, and the evaluation of the kernel network can be viewed as a $n \times n$ matrix K, with its x, y entry $(K)_{xy} = \kappa_\phi(a(x), a(y), x, y)$. Then the action of $\mathcal{K}_a$ becomes the matrix-vector multiplication Ku. For the lifted representation (3.3), for each $x \in D_j$, $v^{(t)}(x) \in \mathbb{R}^{d_v}$ and the output of the kernel $(K)_{xy} = \kappa_\phi(a(x), a(y), x, y)$ is a $d_v \times d_v$ matrix. Therefore K becomes a fourth order tensor with shape $n \times n \times d_v \times d_v$.

Kernel convolution on graphs. Since we assume a non-uniform discretization of D that can differ for each data pair, computing with (3.3) cannot be implemented in a standard way. Graph neural networks offer a natural solution since message passing on graphs can be viewed as the integration (3.2). Since the message passing is computed locally, it avoids storing the full kernel matrix K. Given a discretization D_j, we can adaptively construct graphs on the domain D. The structure of the graph's adjacency matrix transfers to the kernel matrix K. We define the edge attributes $e(x, y) = (a(x), a(y), x, y) \in \mathbb{R}^{2(d+1)}$ and update the graph nodes following (3.3)

which mimics the message passing neural network (Gilmer et al., 2017):

$$v^{(t+1)}(x) = (\sigma(W + \mathcal{K}_a)v^{(t)})(x) \approx \sigma\Big(Wv^{(t)} + \frac{1}{|N(x)|}\sum_{y\in N(x)} \kappa_\phi\big(e(x,y)\big)v^{(t)}(y)\Big) \tag{3.4}$$

where $N(x)$ is the neighborhood of x, in this case, the entire discritized domain D_j.

Domain of Integration. Construction of fully connected graphs is memory intensive and can become computationally infeasible for fine discretizations, i.e., when $|D_j|$ is large. To partially alleviate this, we can ignore the longest range kernel interactions as they have decayed the most and change the integration domain in (3.2) from D to $B(x, r)$ for some fixed radius $r > 0$. This is equivalent to imposing a sparse structure on the kernel matrix K so that only entries around the diagonal are non-zero and results in the complexity $O(n^2 r^d)$.

Nyström approximation. To further relieve computational complexity, we use Nyström approximation or the inducing points method by uniformly sampling $m < n$ nodes from the n nodes discretization, which is to approximate the kernel matrix by a low-rank decomposition

$$K_{nn} \approx K_{nm}K_{mm}K_{mn} \tag{3.5}$$

where $K_{nn} = K$ is the original $n\times n$ kernel matrix and K_{mm} is the $m\times m$ kernel matrix corresponding to the m inducing points. K_{nm} and K_{mn} are transition matrices which could include restriction, prolongation, and interpolation. Nyström approximation further reduces the complexity to $O(m^2 r^d)$.

3.4 Multipole Graph Kernel Network

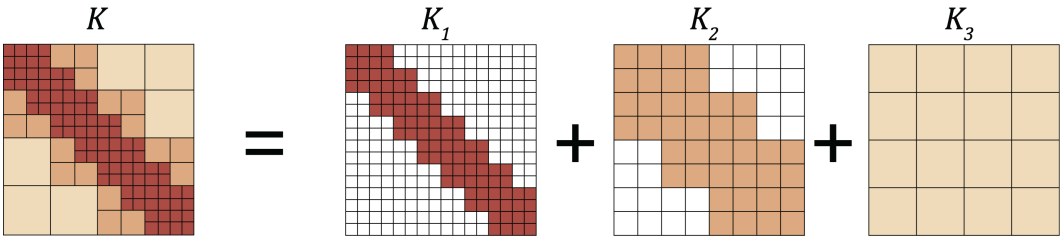
The fast multipole method (FMM) is a systematic approach of combining the aforementioned sparse and low-rank approximations while achieving linear complexity. The kernel matrix is decomposed into different ranges and a hierarchy of low-rank structures is imposed on the long-range components. We employ this idea to construct hierarchical, multi-scale graphs, without being constraint to particular forms of the kernel (L. Ying, Biros, and Zorin, 2004). We elucidate the workings of the FMM through matrix factorization.

The key to the fast multipole method's linear complexity lies in the subdivision of

the kernel matrix according to the range of interaction, as shown in Figure 3.2:

$$K = K_1 + K_2 + \ldots + K_L \tag{3.6}$$

where K_1 corresponds to the shortest-range interaction, and K_L corresponds to the longest-range interaction. While the uniform grids depicted in Figure 3.2 produce an orthogonal decomposition of the kernel, the decomposition may be generalized to arbitrary graphs by allowing slight overlap.



The kernel matrix K is decomposed respect to ranges. K_1 corresponds to short-range interaction; it is sparse but high-rank. K_3 corresponds to long-range interaction; it is dense but low-rank.

Figure 3.2: Hierarchical matrix decomposition

Multi-scale graphs

We construct L graph levels, where the finest graph corresponds to the shortest-range interaction K_1, and the coarsest graph corresponds to the longest-range interaction K_L. In what follows, we will drop the time dependence from (3.3) and use the subscript v_l to denote the representation at each level of the graph. Assuming the underlying graph is a uniform grid with resolution s such that $s^d = n$, the L multi-level graphs will be grids with resolution $s_l = s/2^{l-1}$, and consequentially $n_l = s_l^d = (s/2^{l-1})^d$ for $l = 1, \ldots, L$. In general, the underlying discretization can be arbitrary and we produce a hierarchy of L graphs with a decreasing number of nodes $n_1, \ldots, n_L$.

The coarse graph representation can be understood as recursively applying an inducing points approximation: starting from a graph with $n_1 = n$ nodes, we impose inducing points of size $n_2, n_3, \ldots$ which all admit a low-rank kernel matrix decomposition of the form (3.5). The original $n \times n$ kernel matrix K_l is represented by a much smaller $n_l \times n_l$ kernel matrix, denoted by $K_{l,l}$. As shown in Figure (3.2), K_1 is full-rank but very sparse while K_L is dense but low-rank. Such structure can be achieved by applying equation (3.5) recursively to equation (3.6), leading to the

multi-resolution matrix factorization (Kondor, Teneva, and Garg, 2014):

$$K \approx K_{1,1} + K_{1,2}K_{2,2}K_{2,1} + K_{1,2}K_{2,3}K_{3,3}K_{3,2}K_{2,1} + \cdots \tag{3.7}$$

where $K_{1,1} = K_1$ represents the shortest range, $K_{1,2}K_{2,2}K_{2,1} \approx K_2$, represents the second shortest range, etc. The center matrix $K_{l,l}$ is a $n_l \times n_l$ kernel matrix corresponding to the l-level of the graph described above. The long matrices $K_{l+1,l}, K_{l,l+1}$ are $n_{l+1} \times n_l$ and $n_{l+1} \times n_l$ transition matrices. We define them as moving the representation v_l between different levels of the graph via an integral kernel that we learn. In general, $v^{(t)}(x) \in \mathbb{R}^{d_v}$ and the output of the kernel $(K_{l,l'})_{xy} = \kappa_\phi(a(x), a(y), x, y)$ is itself a $d_v \times d_v$ matrix, so all these matrices are again fourth-order tensors.

$$K_{l,l} : v_l \mapsto v_l = \int_{B(x,r_{l,l})} \kappa_{\phi_{l,l}}(a(x), a(y), x, y) v_l(y)\, dy \tag{3.8}$$

$$K_{l+1,l} : v_l \mapsto v_{l+1} = \int_{B(x,r_{l+1,l})} \kappa_{\phi_{l+1,l}}(a(x), a(y), x, y) v_l(y)\, dy \tag{3.9}$$

$$K_{l,l+1} : v_{l+1} \mapsto v_l = \int_{B(x,r_{l,l+1})} \kappa_{\phi_{l,l+1}}(a(x), a(y), x, y) v_{l+1}(y)\, dy \tag{3.10}$$

Linear complexity. Each matrix in the decomposition (3.6) is represented by the kernel matrix $K_{l,l}$ corresponding to the appropriate sub-graph. Since the number of non-zero entries of each row in these matrices is constant, we obtain that the computational complexity is $\sum_l O(n_l)$. By designing the sub-graphs so that n_l decays fast enough, we can obtain linear complexity. For example, choose $n_l = O(n/2^l)$ then $\sum_l O(n_l) = \sum_l n/2^l = O(n)$. Combined with a Nyström approximation, we obtain $O(m)$ complexity.

V-cycle Algorithm

We present a V-cycle algorithm, see Figure 3.1, for efficiently computing (3.7). It consists of two steps: the **downward pass** and the **upward pass**. Denote the representation in downward pass and upward pass by $\check{v}$ and $\hat{v}$, respectively. In the downward step, the algorithm starts from the fine graph representation $\check{v}_1$ and updates it by applying a downward transition $\check{v}_{l+1} = K_{l+1,l}\check{v}_l$. In the upward step, the algorithm starts from the coarse presentation $\hat{v}_L$ and updates it by applying an upward transition and the center kernel matrix $\hat{v}_l = K_{l,l-1}\hat{v}_{l-1} + K_{l,l}\check{v}_l$. Notice that the one level downward and upward exactly computes $K_{1,1} + K_{1,2}K_{2,2}K_{2,1}$, and a full L-level v-cycle leads to the multi-resolution decomposition (3.7).

Multipole graph kernel network

Employing (3.8)-(3.10), we use L neural networks $\kappa_{\phi_{1,1}}, \ldots, \kappa_{\phi_{L,L}}$ to approximate the kernel $K_{l,l}$, and $2(L-1)$ neural networks $\kappa_{\phi_{1,2}}, \kappa_{\phi_{2,1}}, \ldots$ to approximate the transitions $K_{l+1,l}, K_{l,l+1}$. Following the iterative architecture (3.3), we also introduce the linear operator W, denoting it by W_l for each corresponding resolution. Since it acts on a fixed resolution, we employ it only along with the kernel $K_{l,l}$ and not the transitions. At each time step $t = 0, \ldots, T-1$, we perform a full V-cycle:

Downward Pass:

$$\text{For } l = 1, \ldots, L: \qquad \check{v}_{l+1}^{(t+1)} = \sigma(\hat{v}_{l+1}^{(t)} + K_{l+1,l}\check{v}_{l}^{(t+1)}) \tag{3.11}$$

Upward Pass:

$$\text{For } l = L, \ldots, 1: \qquad \hat{v}_{l}^{(t+1)} = \sigma((W_l + K_{l,l})\check{v}_{l}^{(t+1)} + K_{l,l-1}\hat{v}_{l-1}^{(t+1)}). \tag{3.12}$$

We initialize as $v_1^{(0)} = Pu^{(0)} = Pa$ and output $u^{(T)} = Qv^{(T)} = Q\hat{v}_1^{(T)}$. The algorithm unifies multi-resolution matrix decomposition with iterative graph kernel networks. Combined with a Nyström approximation it leads to $O(m)$ computational complexity that can be implemented with message passing neural networks. Notice GKN is a specific case of V-cycle when $L = 1$.

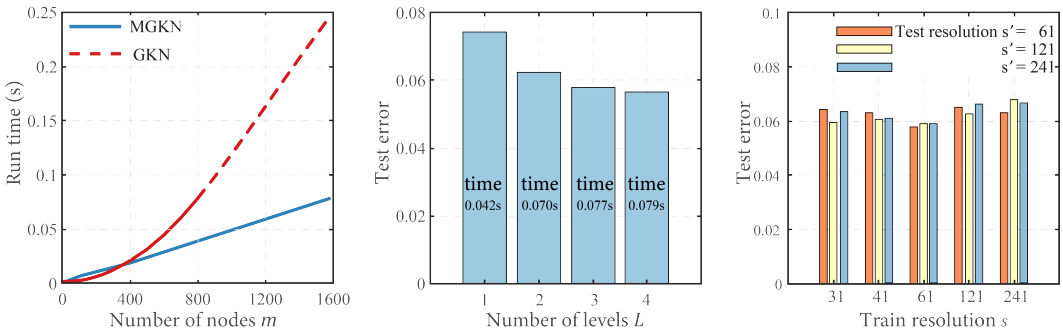
3.5 Experiments

Properties of the multipole graph kernel network

In this section, we show that MGKN has linear complexity and learns discretization invariant solutions by solving the steady-state of Darcy flow. In particular, we consider the 2-d PDE

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x) \qquad && x \in (0,1)^2 \\ u(x) &= 0 && x \in \partial(0,1)^2 \end{aligned} \tag{3.13}$$

and approximate the mapping $a \mapsto u$ which is non-linear despite the fact that (3.13) is a linear PDE. We model the coefficients a as random piece-wise constant functions and generate data by solving (3.13) using a second-order finite difference scheme on a fine grid. Data of coarser resolutions are sub-sampled. See the supplements for further details. The code depends on Pytorch Geometric(Fey and Lenssen, 2019), also included in the supplements.



Left: compared to GKN whose complexity scales quadratically with the number of nodes, MGKN has a linear complexity; **Mid**: Adding more levels reduces test error; **Right**: MGKN can be trained on a coarse resolution and perform well when tested on a fine resolution, showing invariance to discretization.

Figure 3.3: Properties of multipole graph kernel network (MGKN) on Darcy flow

We use Nyström approximation by sampling $m_1, \ldots, m_L$ nodes for each level. When changing the number of levels, we fix coarsest level $m_L = 25, r_{L,L} = 2^{-1}$, and let $m_l = 25 \cdot 4^{L-l}$, $r_{l,l} = 2^{-(L-l)}$, and $r_{l,l+1} = r_{l+1,l} = 2^{-(L-l)+1/2}$. This set-up is one example that can obtain linear complexity. In general, any choice satisfying $\sum_l m_l^2 r_{l,l}^2 = O(m_1)$ also works. We set width $d_v = 64$, iteration $T = 5$ and kernel network $\kappa_{\phi_{l,l}}$ as a three-layer neural network with width $256/2^{(l-1)}$, so that coarser grids will have smaller kernel networks.

1. Linear complexity: The left most plot in Figure 3.3 shows that MGKN (blue line) achieves linear time complexity (the time to evaluate one equation) w.r.t. the number of nodes, while GKN (red line) has quadratic complexity (the solid line is interpolated; the dash line is extrapolated). Since the GPU memory used for backpropagation also scales with the number of edges, GKN is limited to $m \leq 800$ on a single 11G-GPU while MGKN can scale to much higher resolutions . In other words, MGKN can be applicable for larger settings where GKN cannot.

2. Comparing with single-graph: As shown in Figure 3.3 (mid), adding multi-leveled graphs helps decrease the error. The MGKN depicted in blue bars starts from a fine sampling $L = 1; m = [1600]$, and adding subgraphs, $L = 2; m = [400, 1600]$, $L = 3; m = [100, 400, 1600]$, up to, $L = 4; m = [25, 100, 400, 1600]$. When $L = 1$, MGKN and GKN are equivalent. This experiment shows using multi-level graphs helps improve accuracy without increasing much of time-complexity.

3. Generalization to resolution: The MGKN is discretization invariant, and therefore capable of super-resolution. We train with nodes sampled from a $s \times s$ resolution mesh and test on nodes sampled from a $s' \times s'$ resolution mesh. As shown in on the right of Figure 3.3, MGKN achieves similar testing error on $s' = 61, 121, 241$, independently of the training discretization.

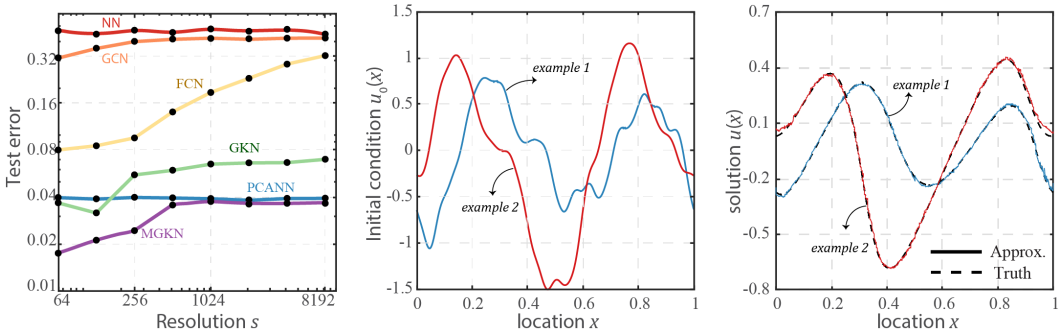
Comparison with benchmarks

We compare the accuracy of our methodology with other deep learning methods as well as reduced order modeling techniques that are commonly used in practice. As a test bed, we use Darcy flow (3.13) and the 1-d viscous Burger's equations:

$$\begin{aligned} \partial_t u(x,t) + \partial_x(u^2(x,t)/2) &= \nu \partial_{xx} u(x,t), \qquad && x \in (0, 2\pi), t \in (0,1] \\ u(x,0) &= u_0(x), && x \in (0, 2\pi) \end{aligned} \tag{3.14}$$

with periodic boundary conditions. We consider mapping the initial condition to the solution at time one $u_0 \mapsto u(\cdot, 1)$. Burger's equation re-arranges low to mid range energies resulting in steep discontinuities that are dampened proportionately to the size of the viscosity ν. It acts as a simplified model for the Navier-Stokes equation. We sample initial conditions as Gaussian random fields and solve (6.18) via a split-step method on a fine mesh, sub-sampling other data as needed; two examples of u_0 and $u(\cdot, 1)$ are shown in the middle and right of Figure 3.4.

Figure 3.4 shows the relative test errors for a variety of methods on Burger's (6.18) (left) as a function of the grid resolution. First notice that MGKN achieves a constant steady state test error, demonstrating that it has learned the true infinite-dimensional solution operator irrespective of the discretization. This is in contrast to the state-of-the-art fully convolution network (**FCN**) proposed in (Zhu and Zabaras, 2018) which has the property that what it learns is tied to a specific discretization. Indeed, we see that, in both cases, the error increases with the resolution since standard convolution layer are parametrized locally and therefore cannot capture the long-range correlations of the solution operator. Using linear spaces, the (**PCA+NN**) method proposed in (Bhattacharya, Kovachki, and Andrew M. Stuart, 2020) utilizes deep learning to produce a fast, fully data-driven reduced order model. The graph convolution network (**GCN**) method follows (Alet et al., 2019)'s architecture, with naive nearest neighbor connection. It shows simple nearest-neighbor graph structures are insufficient. The graph kernel network (**GKN**) employs an architecture similar to (3.3) but, without the multi-level graph extension, it can be slow due the quadratic time complexity. For Burger's equation, when



1-d Burgers equation with viscosity $\nu = 0.1$. **Left**: performance of different methods. MGKN has competitive performance. **Mid**: input functions (u_0) of two examples. **Right**: corresponding outputs from MGKN of the two examples, and their ground truth (dash line). The error is minimal on both examples.

Figure 3.4: Comparsion with benchmarks on Burgers equation, with examples of inputs and outputs.

linear spaces are no longer near-optimal, MGKN is the best performing method. This is a very encouraging result since many challenging applied problem are not well approximated by linear spaces and can therefore greatly benefit from non-linear approximation methods such as MGKN. For details on the other methods, see the supplements.

3.6 Discussion and Conclusion

We introduced the multipole graph kernel network (MGKN), a graph-based algorithm able to capture correlations in data at any length scale with a linear time complexity. Our work ties together ideas from graph neural networks, multi-scale modeling, and matrix factorization. Using a kernel integration architecture, we validate our methodology by showing that it can learn mesh-invariant solutions operators to parametric PDEs. Ideas in this work are not tied to our particular applications and can be used provide significant speed-up for processing densely connected graph data.

References

Abu-El-Haija, Sami, Amol Kapoor, et al. (2018). "N-gcn: Multi-scale graph convolution for semi-supervised node classification". In: *arXiv preprint arXiv:1802.08888*.

Abu-El-Haija, Sami, Bryan Perozzi, et al. (2019). "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing". In: *arXiv preprint arXiv:1905.00067*.

Adler, Jonas and Ozan Oktem (Nov. 2017). "Solving ill-posed inverse problems using iterative deep neural networks". In: *Inverse Problems*. DOI: 10.1088/1361-6420/aa9581. URL: https://doi.org/10.1088%2F1361-6420%2Faa9581.

Alet, Ferran et al. (2019). "Graph Element Networks: adaptive, structured computation and memory". In: *36th International Conference on Machine Learning*. PMLR. URL: http://proceedings.mlr.press/v97/alet19a.html.

Alfke, Dominik and Martin Stoll (2019). "Semi-Supervised Classification on Non-Sparse Graphs Using Low-Rank Graph Convolutional Networks". In: *CoRR* abs/1905.10224. arXiv: 1905.10224. URL: http://arxiv.org/abs/1905.10224.

Bar, Leah and Nir Sochen (2019). "Unsupervised deep learning algorithm for PDE-based forward and inverse problems". In: *arXiv preprint arXiv:1904.05417*.

Bhatnagar, Saakaar et al. (2019). "Prediction of aerodynamic flow fields using convolutional neural networks". In: *Computational Mechanics*, pp. 1–21.

Bhattacharya, Kaushik, Nikola B. Kovachki, and Andrew M. Stuart (2020). "Model Reduction and Neural Networks for Parametric PDE(s)". In: *arXiv preprint arXiv:2005.03180*.

Börm, Steffen, Lars Grasedyck, and Wolfgang Hackbusch (2003). "Hierarchical matrices". In: *Lecture notes* 21, p. 2003.

E, Weinan and Bing Yu (Mar. 2018). "The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems". English (US). In: *Communications in Mathematics and Statistics*. ISSN: 2194-6701. DOI: 10.1007/s40304-018-0127-z.

Fan, Yuwei, Jordi Feliu-Faba, et al. (2019). "A multiscale neural network based on hierarchical nested bases". In: *Research in the Mathematical Sciences* 6.2, p. 21.

Fan, Yuwei, Lin Lin, et al. (2019). "A multiscale neural network based on hierarchical matrices". In: *Multiscale Modeling & Simulation* 17.4, pp. 1189–1213.

Fey, Matthias and Jan E. Lenssen (2019). "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning*.

Gao, Hongyang and Shuiwang Ji (2019). "Graph U-Nets". In: *arXiv preprint arXiv:1905.05178*.

Gilmer, Justin et al. (2017). "Neural message passing for quantum chemistry". In: *Proceedings of the 34th International Conference on Machine Learning*.

Greengard, Leslie and Vladimir Rokhlin (1997). "A new version of the fast multipole method for the Laplace equation in three dimensions". In: *Acta numerica* 6, pp. 229–269.

Guo, Xiaoxiao, Wei Li, and Francesco Iorio (2016). "Convolutional neural networks for steady flow approximation". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Hamilton, Will, Zhitao Ying, and Jure Leskovec (2017). "Inductive representation learning on large graphs". In: *Advances in neural information processing systems*, pp. 1024–1034.

Ithapu, Vamsi K et al. (2017). "The incremental multiresolution matrix factorization algorithm". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2951–2960.

Jiang, Chiyu Max et al. (2020). "MeshfreeFlowNet: A Physics-Constrained Deep Continuous Space-Time Super-Resolution Framework". In: *arXiv preprint arXiv:2005.01463*.

Kipf, Thomas N and Max Welling (2016). "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907*.

Kondor, Risi, Nedelina Teneva, and Vikas Garg (2014). "Multiresolution matrix factorization". In: *International Conference on Machine Learning*, pp. 1620–1628.

Li, Maosen et al. (2020). "Dynamic Multiscale Graph Neural Networks for 3D Skeleton-Based Human Motion Prediction". In: *arXiv preprint arXiv:2003.08802*.

Li, Zongyi et al. (2020). "Neural operator: Graph kernel network for partial differential equations". In: *arXiv preprint arXiv:2003.03485*.

Lu, Lu, Pengzhan Jin, and George Em Karniadakis (2019). "DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators". In: *arXiv preprint arXiv:1910.03193*.

Murphy, Ryan L et al. (2018). "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs". In: *arXiv preprint arXiv:1811.01900*.

Nelsen, Nicholas H and Andrew M Stuart (2020). "The Random Feature Model for Input-Output Maps between Banach Spaces". In: *arXiv preprint arXiv:2005.10224*.

Raissi, Maziar, Paris Perdikaris, and George E Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378, pp. 686–707.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241.

Rozza, Gianluigi, Dinh Bao Phuong Huynh, and Anthony T Patera (2007). "Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations". In: *Archives of Computational Methods in Engineering* 15.3, p. 1.

Sauter, Stefan A. and Christoph Schwab (2010). *Boundary Element Methods*. Springer Series in Computational Mathematics.

Smith, Jonathan D, Kamyar Azizzadenesheli, and Zachary E Ross (2020). "Eikonet: Solving the eikonal equation with deep neural networks". In: *arXiv preprint arXiv:2004.00361*.

Su, Chau Hsing and Clifford S Gardner (1969). "Korteweg-de Vries equation and generalizations. III. Derivation of the Korteweg-de Vries equation and Burgers equation". In: *Journal of Mathematical Physics* 10.3, pp. 536–539.

Tek, MR et al. (1957). "Development of a generalized Darcy equation". In: *Journal of Petroleum Technology* 9.06, pp. 45–47.

Ummenhofer, Benjamin et al. (2020). "Lagrangian fluid simulation with continuous convolutions". In: *International Conference on Learning Representations*.

Veličković, Petar et al. (2017). "Graph attention networks". In.

Wu, Zonghan et al. (2020). "A comprehensive survey on graph neural networks". In: *IEEE Transactions on Neural Networks and Learning Systems*.

Ying, Lexing, George Biros, and Denis Zorin (2004). "A kernel-independent adaptive fast multipole algorithm in two and three dimensions". In: *Journal of Computational Physics* 196.2, pp. 591–626.

Zhu, Yinhao and Nicholas Zabaras (2018). "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification". In: *Journal of Computational Physics*. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.04.018. URL: http://www.sciencedirect.com/science/article/pii/S0021999118302341.

Zienkiewicz, Olgierd Cecil et al. (1977). *The finite element method*. Vol. 3. McGraw-hill London.

Chapter 4

MODEL: FOURIER NEURAL OPERATOR

The classical development of neural networks has primarily focused on learning mappings between finite-dimensional Euclidean spaces. Recently, this has been generalized to neural operators that learn mappings between function spaces. For partial differential equations (PDEs), neural operators directly learn the mapping from any functional parametric dependence to the solution. Thus, they learn an entire family of PDEs, in contrast to classical methods which solve one instance of the equation. In this work, we formulate a new neural operator by parameterizing the integral kernel directly in Fourier space, allowing for an expressive and efficient architecture. We perform experiments on Burgers' equation, Darcy flow, and Navier-Stokes equation. Experiments shows that Fourier Neural Operator achieves superior accuracy compared to previous learning-based solvers under fixed resolution.

4.1 Introduction

Many problems in science and engineering involve solving complex partial differential equation (PDE) systems repeatedly for different values of some parameters. Examples arise in molecular dynamics, micro-mechanics, and turbulent flows. Often such systems require fine discretization in order to capture the phenomenon being modeled. As a consequence, traditional numerical solvers are slow and sometimes inefficient. For example, when designing materials such as airfoils, one needs to solve the associated inverse problem where thousands of evaluations of the forward model are needed. A fast method can make such problems feasible.

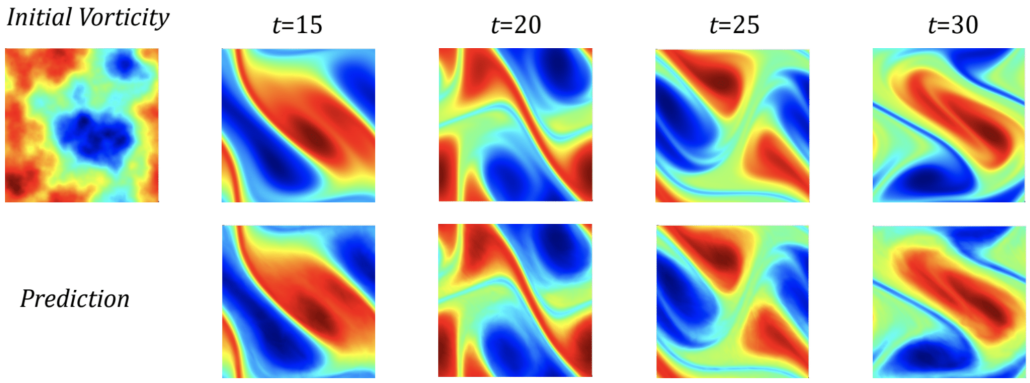
Conventional solvers vs. Data-driven methods. Traditional solvers such as finite element methods (FEM) and finite difference methods (FDM) solve the equation by discretizing the space. Therefore, they impose a trade-off on the resolution: coarse grids are fast but less accurate; fine grids are accurate but slow. Complex PDE systems, as described above, usually require a very fine discretization, and therefore very challenging and time-consuming for traditional solvers. On the other hand, data-driven methods can directly learn the trajectory of the family of equations from the data. As a result, the learning-based method can be orders of magnitude faster than the conventional solvers.

Machine learning methods may hold the key to revolutionizing scientific disciplines by providing fast solvers that approximate or enhance traditional ones (Raissi, Perdikaris, and George E Karniadakis, 2019; C. M. Jiang et al., 2020; Greenfeld et al., 2019; Kochkov et al., 2021). However, classical neural networks map between finite-dimensional spaces and can therefore only learn solutions tied to a specific discretization. This is often a limitation for practical applications and therefore the development of mesh-invariant neural networks is required. We first outline two mainstream neural network-based approaches for PDEs: the finite-dimensional operators and physics-informed neural network.

Finite-dimensional operators. These approaches parameterize the solution operator as a deep convolutional neural network between finite-dimensional Euclidean spaces Guo, W. Li, and Iorio, 2016; Zhu and Zabaras, 2018; Adler and Oktem, 2017; Bhatnagar et al., 2019; Khoo, J. Lu, and Ying, 2017. Such approaches are, by definition, mesh-dependent and will need modifications and tuning for different resolutions and discretizations in order to achieve consistent error (if at all possible). Furthermore, these approaches are limited to the discretization size and geometry of the training data and hence, it is not possible to query solutions at new points in the domain. In contrast, we show, for our method, both invariance of the error to grid resolution, and the ability to transfer the solution between meshes.

Physics-Informed Neural Network. The second approach directly parameterizes the solution function as a neural network (E and Yu, 2018; Raissi, Perdikaris, and George E Karniadakis, 2019; Bar and Sochen, 2019; Smith, Azizzadenesheli, and Ross, 2020; Pan and Duraisamy, 2020). This approach is designed to model one specific instance of the PDE, not the solution operator. It is mesh-independent and accurate, but for any given new instance of the functional parameter/coefficient, it requires training a new neural network. The approach closely resembles classical methods such as finite elements, replacing the linear span of a finite set of local basis functions with the space of neural networks. The Neural-FEM approach suffers from the same computational issue as classical methods: the optimization problem needs to be solved for every new instance. Furthermore, the approach is limited to a setting in which the underlying PDE is known.

Neural Operators. Recently, a new line of work proposed learning mesh-free, infinite-dimensional operators with neural networks (L. Lu, Jin, and George Em



Zero-shot super-resolution: Navier-Stokes Equation with viscosity ν = 1e−4; Ground truth on top and prediction on bottom; trained on $64\times64\times20$ dataset; evaluated on $256\times256\times80$ (see Section 4.4).

Figure 4.1: Zero-shot super-resolution on Navier-Stokes.

Karniadakis, 2019; Bhattacharya, Kovachki, and Andrew M. Stuart, 2020; Nelsen and A. Stuart, 2020; Z. Li et al., 2020b; Z. Li et al., 2020a; Patel et al., 2021). The neural operator remedies the mesh-dependent nature of the finite-dimensional operator methods discussed above by producing a single set of network parameters that may be used with different discretizations. It has the ability to transfer solutions between meshes. Furthermore, the neural operator needs to be trained only once. Obtaining a solution for a new instance of the parameter requires only a forward pass of the network, alleviating the major computational issues incurred in Neural-FEM methods. Lastly, the neural operator requires no knowledge of the underlying PDE, only data. Thus far, neural operators have not yielded efficient numerical algorithms that can parallel the success of convolutional or recurrent neural networks in the finite-dimensional setting due to the cost of evaluating integral operators. Through the fast Fourier transform, our work alleviates this issue.

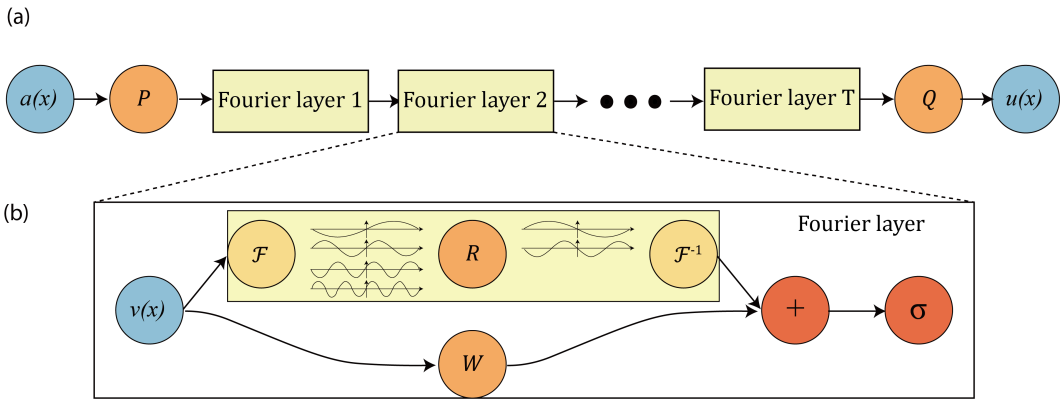
Fourier Transform. The Fourier transform is frequently used in spectral methods for solving differential equations, since differentiation is equivalent to multiplication in the Fourier domain. Fourier transforms have also played an important role in the development of deep learning. In theory, they appear in the proof of the universal approximation theorem (Hornik, Stinchcombe, White, et al., 1989) and, empirically, they have been used to speed up convolutional neural networks (Mathieu, Henaff, and LeCun, 2013). Neural network architectures involving the Fourier transform or the use of sinusoidal activation functions have also been proposed and studied (**mingo2004Fourier**; Bengio, LeCun, et al., 2007; Sitzmann et al., 2020). Recently,

some spectral methods for PDEs have been extended to neural networks (Fan, Bohorquez, and Ying, 2019; Fan, Lin, et al., 2019; Kashinath, Marcus, et al., 2020). We build on these works by proposing a neural operator architecture defined directly in Fourier space with quasi-linear time complexity and state-of-the-art approximation capabilities.

Our Contributions. We introduce the Fourier neural operator, a novel deep learning architecture able to learn mappings between infinite-dimensional spaces of functions; the integral operator is restricted to a convolution, and instantiated through a linear transformation in the Fourier domain.

- The Fourier neural operator learns the resolution-invariant solution operator for the family of Navier-Stokes equation in the turbulent regime, where previous graph-based neural operators do not converge.
- By construction, the method shares the same learned network parameters irrespective of the discretization used on the input and output spaces. It can do zero-shot super-resolution: trained on a lower resolution directly evaluated on a higher resolution, as shown in Figure 4.1.
- The proposed method consistently outperforms all existing deep learning methods even when fixing the resolution to be 64×64. It achieves error rates that are 30% lower on Burgers' Equation, 60% lower on Darcy Flow, and 30% lower on Navier Stokes (turbulent regime with viscosity $\nu = 1e{-}4$). When learning the mapping for the entire time series, the method achieves $< 1\%$ error with viscosity $\nu = 1e{-}3$ and 8% error with viscosity $\nu = 1e{-}4$.
- On a 256×256 grid, the Fourier neural operator has an inference time of only 0.005s compared to the $2.2s$ of the pseudo-spectral method used to solve Navier-Stokes. Despite its tremendous speed advantage, the method does not suffer from accuracy degradation when used in downstream applications such as solving the Bayesian inverse problem, as shown in Figure 4.4.

We observed that the proposed framework can approximate complex operators raising in PDEs that are highly non-linear, with high frequency modes and slow energy decay. The power of neural operators comes from combining linear, global integral operators (via the Fourier transform) and non-linear, local activation functions. Similar to the way standard neural networks approximate highly non-linear functions by



(a) The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network P. 2. Apply four layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u. **(b) Fourier layers**: Start from input v. On top: apply the Fourier transform $\mathcal{G}$; a linear transform R on the lower Fourier modes and filters out the higher modes; then apply the inverse Fourier transform $\mathcal{G}^{-1}$. On the bottom: apply a local linear transform W.

Figure 4.2: The architecture of the neural operators and Fourier layer

combining linear multiplications with non-linear activations, the proposed neural operators can approximate highly non-linear operators.

4.2 Neural Operator

The neural operator, proposed in (Z. Li et al., 2020b), is formulated as an iterative architecture $v_0 \mapsto v_1 \mapsto \ldots \mapsto v_T$ where v_j for $j = 0, 1, \ldots, T-1$ is a sequence of functions each taking values in $\mathbb{R}^{d_v}$. As shown in Figure 4.2 (a), the input $a \in \mathcal{A}$ is first lifted to a higher dimensional representation $v_0(x) = P(a(x))$ by the local transformation P which is usually parameterized by a shallow fully-connected neural network. Then we apply several iterations of updates $v_t \mapsto v_{t+1}$ (defined below). The output $u(x) = Q(v_T(x))$ is the projection of v_T by the local transformation $Q : \mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$. In each iteration, the update $v_t \mapsto v_{t+1}$ is defined as the composition of a non-local integral operator $\mathcal{K}$ and a local, nonlinear activation function σ.

Definition 4.2.1 (Iterative updates) *Define the update to the representation* $v_t \mapsto v_{t+1}$ *by*

$$v_{t+1}(x) := \sigma\Big(W v_t(x) + \big(\mathcal{K}(a;\phi)v_t\big)(x)\Big), \qquad \forall x \in D \tag{4.1}$$

where $\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}} \to \mathcal{L}(\mathcal{U}(D;\mathbb{R}^{d_v}), \mathcal{U}(D;\mathbb{R}^{d_v}))$ *maps to bounded linear operators on* $\mathcal{U}(D;\mathbb{R}^{d_v})$ *and is parameterized by* $\phi \in \Theta_{\mathcal{K}}$, $W : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$ *is a linear transformation, and* $\sigma : \mathbb{R} \to \mathbb{R}$ *is a non-linear activation function whose action is defined component-wise.*

We choose $\mathcal{K}(a;\phi)$ to be a kernel integral transformation parameterized by a neural network.

Definition 4.2.2 (Kernel integral operator $\mathcal{K}$) *Define the kernel integral operator mapping in (4.1) by*

$$\big(\mathcal{K}(a;\phi)v_t\big)(x) := \int_D \kappa\big(x, y, a(x), a(y);\phi\big)v_t(y)\mathrm{d}y, \qquad \forall x \in D \tag{4.2}$$

where $\kappa_\phi : \mathbb{R}^{2(d+d_a)} \to \mathbb{R}^{d_v \times d_v}$ is a neural network parameterized by $\phi \in \Theta_{\mathcal{K}}$.

Here κ_ϕ plays the role of a kernel function which we learn from data. Together definitions 1 and 2 constitute a generalization of neural networks to infinite-dimensional spaces as first proposed in Z. Li et al., 2020b. Notice even the integral operator is linear, the neural operator can learn highly non-linear operators by composing linear integral operators with non-linear activation functions, analogous to standard neural networks.

If we remove the dependence on the function a and impose $\kappa_\phi(x, y) = \kappa_\phi(x - y)$, we obtain that (4.2) is a convolution operator, which is a natural choice from the perspective of fundamental solutions. We exploit this fact in the following section by parameterizing κ_ϕ directly in Fourier space and using the Fast Fourier Transform (FFT) to efficiently compute (4.2). This leads to a fast architecture that obtains state-of-the-art results for PDE problems.

4.3 Fourier Neural Operator

We propose replacing the kernel integral operator in (4.2), by a convolution operator defined in Fourier space. Let $\mathcal{G}$ denote the Fourier transform of a function $f : D \to \mathbb{R}^{d_v}$ and $\mathcal{G}^{-1}$ its inverse then

$$(\mathcal{G}f)_j(k) = \int_D f_j(x)e^{-2i\pi\langle x,k\rangle}\mathrm{d}x, \qquad (\mathcal{G}^{-1}f)_j(x) = \int_D f_j(k)e^{2i\pi\langle x,k\rangle}\mathrm{d}k$$

for $j = 1, \ldots, d_v$ where $i = \sqrt{-1}$ is the imaginary unit. By letting $\kappa_\phi(x, y, a(x), a(y)) = \kappa_\phi(x - y)$ in (4.2) and applying the convolution theorem, we find that

$$\big(\mathcal{K}(a;\phi)v_t\big)(x) = \mathcal{G}^{-1}\big(\mathcal{G}(\kappa_\phi) \cdot \mathcal{G}(v_t)\big)(x), \qquad \forall x \in D.$$

We, therefore, propose to directly parameterize κ_ϕ in Fourier space.

Definition 4.3.1 (Fourier integral operator $\mathcal{K}$) *Define the Fourier integral operator*

$$\big(\mathcal{K}(\phi)v_t\big)(x) = \mathcal{G}^{-1}\Big(R_\phi \cdot (\mathcal{G}v_t)\Big)(x) \qquad \forall x \in D \tag{4.3}$$

where R_ϕ is the Fourier transform of a periodic function $\kappa : \bar{D} \to \mathbb{R}^{d_v \times d_v}$ parameterized by $\phi \in \Theta_{\mathcal{K}}$. An illustration is given in Figure 4.2 (b).

For frequency mode $k \in D$, we have $(\mathcal{G}v_t)(k) \in C^{d_v}$ and $R_\phi(k) \in C^{d_v \times d_v}$. Notice that since we assume κ is periodic, it admits a Fourier series expansion, so we may work with the discrete modes $k \in \mathbb{Z}^d$. We pick a finite-dimensional parameterization by truncating the Fourier series at a maximal number of modes $k_{\max} = |Z_{k_{\max}}| = |\{k \in \mathbb{Z}^d : |k_j| \le k_{\max,j}, \text{ for } j = 1, \dots, d\}|$. We thus parameterize R_ϕ directly as complex-valued $(k_{\max} \times d_v \times d_v)$-tensor comprising a collection of truncated Fourier modes and therefore drop ϕ from our notation. Since κ is real-valued, we impose conjugate symmetry. We note that the set $Z_{k_{\max}}$ is not the canonical choice for the low frequency modes of v_t. Indeed, the low frequency modes are usually defined by placing an upper-bound on the ℓ_1-norm of $k \in \mathbb{Z}^d$. We choose $Z_{k_{\max}}$ as above since it allows for an efficient implementation.

The discrete case and the FFT. Assuming the domain D is discretized with $n \in \mathbb{N}$ points, we have that $v_t \in \mathbb{R}^{n \times d_v}$ and $\mathcal{G}(v_t) \in C^{n \times d_v}$. Since we convolve v_t with a function which only has $k_{\max}$ Fourier modes, we may simply truncate the higher modes to obtain $\mathcal{G}(v_t) \in C^{k_{\max} \times d_v}$. Multiplication by the weight tensor $R \in C^{k_{\max} \times d_v \times d_v}$ is then

$$\big(R \cdot (\mathcal{G}v_t)\big)_{k,l} = \sum_{j=1}^{d_v} R_{k,l,j} (\mathcal{G}v_t)_{k,j}, \qquad k = 1, \dots, k_{\max}, \quad j = 1, \dots, d_v. \tag{4.4}$$

When the discretization is uniform with resolution $s_1 \times \cdots \times s_d = n$, $\mathcal{G}$ can be replaced by the Fast Fourier Transform. For $f \in \mathbb{R}^{n \times d_v}$, $k = (k_1, \dots, k_d) \in \mathbb{Z}_{s_1} \times \cdots \times \mathbb{Z}_{s_d}$, and $x = (x_1, \dots, x_d) \in D$, the FFT $\hat{\mathcal{G}}$ and its inverse $\hat{\mathcal{G}}^{-1}$ are defined as

$$(\hat{\mathcal{G}} f)_l(k) = \sum_{x_1=0}^{s_1-1} \cdots \sum_{x_d=0}^{s_d-1} f_l(x_1, \dots, x_d) e^{-2i\pi \sum_{j=1}^{d} \frac{x_j k_j}{s_j}},$$

$$(\hat{\mathcal{G}}^{-1} f)_l(x) = \sum_{k_1=0}^{s_1-1} \cdots \sum_{k_d=0}^{s_d-1} f_l(k_1, \dots, k_d) e^{2i\pi \sum_{j=1}^{d} \frac{x_j k_j}{s_j}}$$

for $l = 1, \dots, d_v$. In this case, the set of truncated modes becomes

$$Z_{k_{\max}} = \{(k_1, \dots, k_d) \in \mathbb{Z}_{s_1} \times \cdots \times \mathbb{Z}_{s_d} \mid k_j \le k_{\max,j} \text{ or } s_j - k_j \le k_{\max,j}, \text{ for } j = 1, \dots, d\}.$$

When implemented, R is treated as a $(s_1 \times \cdots \times s_d \times d_v \times d_v)$-tensor and the above definition of $Z_{k_{\max}}$ corresponds to the "corners" of R, which allows for a straight-forward parallel implementation of (4.4) via matrix-vector multiplication. In practice, we have found that choosing $k_{\max,j} = 12$ which yields $k_{\max} = 12^d$ parameters per channel to be sufficient for all the tasks that we consider.

Parameterizations of R. In general, R can be defined to depend on $(\mathcal{G}a)$ to parallel (4.2). Indeed, we can define $R_\phi : \mathbb{Z}^d \times \mathbb{R}^{d_v} \to \mathbb{R}^{d_v \times d_v}$ as a parametric function that maps $\big(k, (\mathcal{G}a)(k)\big)$ to the values of the appropriate Fourier modes. We have experimented with linear as well as neural network parameterizations of R_ϕ. We find that the linear parameterization has a similar performance to the previously described direct parameterization, while neural networks have worse performance. This is likely due to the discrete structure of the space $\mathbb{Z}^d$. Our experiments in this work focus on the direct parameterization presented above.

Invariance to discretization. The Fourier layers are discretization-invariant because they can learn from and evaluate functions which are discretized in an arbitrary way. Since parameters are learned directly in Fourier space, resolving the functions in physical space simply amounts to projecting on the basis $e^{2\pi i \langle x,k \rangle}$ which are well-defined everywhere on $\mathbb{R}^d$. This allows us to achieve zero-shot super-resolution as shown in Section 4.4. Furthermore, our architecture has a consistent error at any resolution of the inputs and outputs. On the other hand, notice that, in Figure 4.3, the standard CNN methods we compare against have an error that grows with the resolution.

Quasi-linear complexity. The weight tensor R contains $k_{\max} < n$ modes, so the inner multiplication has complexity $O(k_{\max})$. Therefore, the majority of the computational cost lies in computing the Fourier transform $\mathcal{G}(v_t)$ and its inverse. General Fourier transforms have complexity $O(n^2)$, however, since we truncate the series the complexity is in fact $O(nk_{\max})$, while the FFT has complexity $O(n \log n)$. Generally, we have found using FFTs to be very efficient. However a uniform discretization is required.

4.4 Numerical Experiments

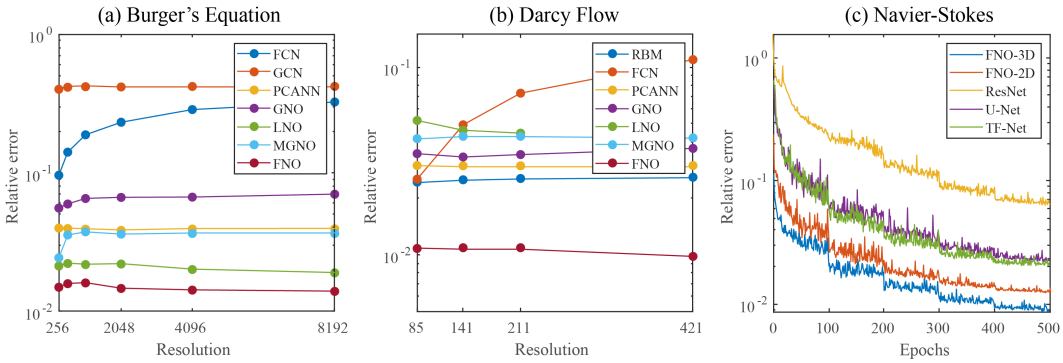
In this section, we compare the proposed Fourier neural operator with multiple finite-dimensional architectures as well as operator-based approximation methods

on the 1-d Burgers' equation, the 2-d Darcy Flow problem, and 2-d Navier-Stokes equation. We do not compare against traditional solvers (FEM/FDM) or neural-FEM type methods since our goal is to produce an efficient operator approximation that can be used for downstream applications. We demonstrate one such application to the Bayesian inverse problem in Section 4.4.

We construct our Fourier neural operator by stacking four Fourier integral operator layers as specified in (4.1) and (9.5) with the ReLU activation as well as batch normalization. Unless otherwise specified, we use $N = 1000$ training instances and 200 testing instances. We use Adam optimizer to train for 500 epochs with an initial learning rate of 0.001 that is halved every 100 epochs. We set $k_{\max,j} = 16, d_v = 64$ for the 1-d problem and $k_{\max,j} = 12, d_v = 32$ for the 2-d problems. Lower resolution data are downsampled from higher resolution. All the computation is carried on a single Nvidia V100 GPU with 16GB memory.

Remark on Resolution. Traditional PDE solvers such as FEM and FDM approximate a single function and therefore their error to the continuum decreases as the resolution is increased. On the other hand, operator approximation is independent of the ways its data is discretized as long as all relevant information is resolved. Resolution-invariant operators have consistent error rates among different resolutions as shown in Figure 4.3. Further, resolution-invariant operators can do zero-shot super-resolution, as shown in Section 4.4.

Benchmarks for time-independent problems (Burgers and Darcy): **NN:** a simple point-wise feedforward neural network. **RBM:** the classical Reduced Basis Method (using a POD basis) (DeVore, 2014). **FCN:** a the-state-of-the-art neural network architecture based on Fully Convolution Networks (Zhu and Zabaras, 2018). **PCANN:** an operator method using PCA as an autoencoder on both the input and output data and interpolating the latent spaces with a neural network (Bhattacharya, Kovachki, and Andrew M. Stuart, 2020). **GNO:** the original graph neural operator (Z. Li et al., 2020b). **MGNO:** the multipole graph neural operator (Z. Li et al., 2020a). **LNO:** a neural operator method based on the low-rank decomposition of the kernel $\kappa(x, y) := \sum_{j=1}^{r} \phi_j(x)\psi_j(y)$, similar to the unstacked DeepONet proposed in (L. Lu, Jin, and George Em Karniadakis, 2019). **FNO:** the newly purposed Fourier neural operator.



Left: benchmarks on Burgers equation; **Mid:** benchmarks on Darcy Flow for different resolutions; **Right:** the learning curves on Navier-Stokes $\nu = 1e{-}3$ with different benchmarks. Train and test on the same resolution.

Figure 4.3: Benchmark on Burger's equation, Darcy Flow, and Navier-Stokes

Benchmarks for time-dependent problems (Navier-Stokes): **ResNet:** 18 layers of 2-d convolution with residual connections (He et al., 2016). **U-Net:** A popular choice for image-to-image regression tasks consisting of four blocks with 2-d convolutions and deconvolutions (Ronneberger, Fischer, and Brox, 2015). **TF-Net:** A network designed for learning turbulent flows based on a combination of spatial and temporal convolutions (Wang et al., 2020). **FNO-2d:** 2-d Fourier neural operator with a RNN structure in time. **FNO-3d:** 3-d Fourier neural operator that directly convolves in space-time.

Burgers' Equation

The 1-d Burgers' equation is a non-linear PDE with various applications including modeling the one dimensional flow of a viscous fluid. It takes the form

$$\begin{aligned} \partial_t u(x,t) + \partial_x(u^2(x,t)/2) &= \nu \partial_{xx} u(x,t), \qquad && x \in (0,1), t \in (0,1] \\ u(x,0) &= u_0(x), && x \in (0,1) \end{aligned} \tag{4.5}$$

with periodic boundary conditions where $u_0 \in L^2_{\text{per}}((0,1);\mathbb{R})$ is the initial condition and $\nu \in \mathbb{R}_+$ is the viscosity coefficient. We aim to learn the operator mapping the initial condition to the solution at time one, $\mathcal{G}^\dagger : L^2_{\text{per}}((0,1);\mathbb{R}) \to H^r_{\text{per}}((0,1);\mathbb{R})$ defined by $u_0 \mapsto u(\cdot,1)$ for any $r > 0$.

The results of our experiments are shown in Figure 4.3 (a) and Table 4.1. Our proposed method obtains the lowest relative error compared to any of the benchmarks. Further, the error is invariant with the resolution, while the error of convolution neural network based methods (FCN) grows with the resolution. Compared to other neural operator methods such as GNO and MGNO that use Nyström sam-

pling in physical space, the Fourier neural operator is both more accurate and more computationally efficient.

Networks	$s = 256$	$s = 512$	$s = 1024$	$s = 2048$	$s = 4096$	$s = 8192$
NN	0.4714	0.4561	0.4803	0.4645	0.4779	0.4452
GCN	0.3999	0.4138	0.4176	0.4157	0.4191	0.4198
FCN	0.0958	0.1407	0.1877	0.2313	0.2855	0.3238
PCANN	0.0398	0.0395	0.0391	0.0383	0.0392	0.0393
GNO	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
FNO	**0.0149**	**0.0158**	**0.0160**	**0.0146**	**0.0142**	**0.0139**

Table 4.1: Benchmarks on 1-d Burgers' equation

Networks	$s = 85$	$s = 141$	$s = 211$	$s = 421$
NN	0.1716	0.1716	0.1716	0.1716
FCN	0.0253	0.0493	0.0727	0.1097
PCANN	0.0299	0.0298	0.0298	0.0299
RBM	0.0244	0.0251	0.0255	0.0259
GNO	0.0346	0.0332	0.0342	0.0369
LNO	0.0520	0.0461	0.0445	–
MGNO	0.0416	0.0428	0.0428	0.0420
FNO	**0.0108**	**0.0109**	**0.0109**	**0.0098**

Table 4.2: Benchmarks on 2-d Darcy Flow

Darcy Flow

We consider the steady-state of the 2-d Darcy Flow equation on the unit box which is the second order, linear, elliptic PDE

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x) \qquad && x \in (0,1)^2 \\ u(x) &= 0 && x \in \partial(0,1)^2 \end{aligned} \tag{4.6}$$

with a Dirichlet boundary where $a \in L^\infty((0,1)^2; \mathbb{R}_+)$ is the diffusion coefficient and $f \in L^2((0,1)^2; \mathbb{R})$ is the forcing function. This PDE has numerous applications including modeling the pressure of subsurface flow, the deformation of linearly elastic materials, and the electric potential in conductive materials. We are interested in learning the operator mapping the diffusion coefficient to the solution, $\mathcal{G}^\dagger : L^\infty((0,1)^2; \mathbb{R}_+) \to H_0^1((0,1)^2; \mathbb{R}_+)$ defined by $a \mapsto u$. Note that although the PDE is linear, the operator $\mathcal{G}^\dagger$ is not.

The results of our experiments are shown in Figure 4.3 (b) and Table 4.2. The proposed Fourier neural operator obtains nearly one order of magnitude lower relative error compared to any benchmarks. We again observe the invariance of the error with respect to the resolution.

2D and 3D Convolutions. FNO-2D, U-Net, TF-Net, and ResNet all do 2D-convolution in the spatial domain and recurrently propagate in the time domain (2D+RNN). The operator maps the solution at the previous 10 time steps to the next time step (2D functions to 2D functions). On the other hand, FNO-3D performs convolution in space-time. It maps the initial time steps directly to the full trajectory (3D functions to 3D functions). The 2D+RNN structure can propagate the solution to any arbitrary time T in increments of a fixed interval length Δt, while the Conv3D structure is fixed to the interval $[0, T]$ but can transfer the solution to an arbitrary time-discretization. We find the 3-d method to be more expressive and easier to train compared to its RNN-structured counterpart.

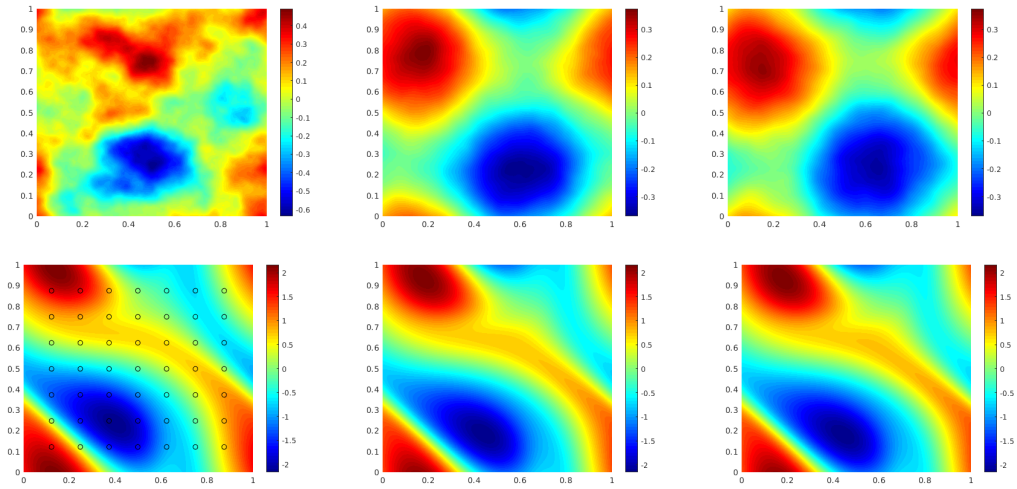
Zero-shot super-resolution.

The neural operator is mesh-invariant, so it can be trained on a lower resolution and evaluated at a higher resolution, without seeing any higher resolution data (zero-shot super-resolution). Figure 4.1 shows an example where we train the FNO-3D model on $64 \times 64 \times 20$ resolution data in the setting above with ($\nu = 1\text{e}{-4}, N = 10000$) and transfer to $256 \times 256 \times 80$ resolution, demonstrating super-resolution in space-time. Fourier neural operator is the only model among the benchmarks (FNO-2D, U-Net, TF-Net, and ResNet) that can do zero-shot super-resolution. And surprisingly, it can do super-resolution not only in the spatial domain but also in the temporal domain.

Bayesian Inverse Problem

In this experiment, we use a function space Markov chain Monte Carlo (MCMC) method (Cotter et al., 2013) to draw samples from the posterior distribution of the initial vorticity in Navier-Stokes given sparse, noisy observations at time $T = 50$. We compare the Fourier neural operator acting as a surrogate model with the traditional solvers used to generate our train-test data (both run on GPU). We generate 25,000 samples from the posterior (with a 5,000 sample burn-in period), requiring 30,000 evaluations of the forward operator.

As shown in Figure 4.4, FNO and the traditional solver recover almost the same posterior mean which, when pushed forward, recovers well the late-time dynamic



The top left panel shows the true initial vorticity while bottom left panel shows the true observed vorticity at $T = 50$ with black dots indicating the locations of the observation points placed on a 7×7 grid. The top middle panel shows the posterior mean of the initial vorticity given the noisy observations estimated with MCMC using the traditional solver, while the top right panel shows the same thing but using FNO as a surrogate model. The bottom middle and right panels show the vorticity at $T = 50$ when the respective approximate posterior means are used as initial conditions.

Figure 4.4: Results of the Bayesian inverse problem for the Navier-Stokes equation.

of Navier Stokes. In sharp contrast, FNO takes $0.005s$ to evaluate a single instance while the traditional solver, after being optimized to use the largest possible internal time-step which does not lead to blow-up, takes $2.2s$. This amounts to 2.5 minutes for the MCMC using FNO and over 18 hours for the traditional solver. Even if we account for data generation and training time (offline steps) which take 12 hours, using FNO is still faster! Once trained, FNO can be used to quickly perform multiple MCMC runs for different initial conditions and observations, while the traditional solver will take 18 hours for every instance. Furthermore, since FNO is differentiable, it can easily be applied to PDE-constrained optimization problems without the need for the adjoint method.

Spectral analysis. Due to the way we parameterize R_ϕ, the function output by (9.5) has at most $k_{\max,j}$ Fourier modes per channel. This, however, does not mean that the Fourier neural operator can only approximate functions up to $k_{\max,j}$ modes. Indeed, the activation functions which occur between integral operators and the final decoder network Q recover the high frequency modes. As an example, consider a solution to the Navier-Stokes equation with viscosity $\nu = 1\text{e}{-3}$. Truncating this

function at 20 Fourier modes yields an error around 2% while our Fourier neural operator learns the parametric dependence and produces approximations to an error of $\leq 1\%$ with only $k_{\max,j} = 12$ parameterized modes.

Non-periodic boundary condition. Traditional Fourier methods work only with periodic boundary conditions. However, the Fourier neural operator does not have this limitation. This is due to the linear transform W (the bias term) which keeps the track of non-periodic boundary. As an example, the Darcy Flow and the time domain of Navier-Stokes have non-periodic boundary conditions, and the Fourier neural operator still learns the solution operator with excellent accuracy.

4.5 Discussion and Conclusion

Requirements on Data. Data-driven methods rely on the quality and quantity of data. To learn Navier-Stokes equation with viscosity $\nu = 1\text{e}{-4}$, we need to generate $N = 10000$ training pairs $\{a_j, u_j\}$ with the numerical solver. However, for more challenging PDEs, generating a few training samples can be already very expensive. A future direction is to combine neural operators with numerical solvers to levitate the requirements on data. **Recurrent structure.** The neural operator has an iterative structure that can naturally be formulated as a recurrent network where all layers share the same parameters without sacrificing performance. (We did not impose this restriction in the experiments.) **Computer vision.** Operator learning is not restricted to PDEs. Images can naturally be viewed as real-valued functions on 2-d domains and videos simply add a temporal structure. Our approach is therefore a natural choice for problems in computer vision where invariance to discretization crucial is important (Chi, B. Jiang, and Mu, 2020).

References

Adler, Jonas and Ozan Oktem (Nov. 2017). "Solving ill-posed inverse problems using iterative deep neural networks". In: *Inverse Problems*. DOI: 10.1088/1361-6420/aa9581. URL: https://doi.org/10.1088%2F1361-6420%2Faa9581.

Bar, Leah and Nir Sochen (2019). "Unsupervised deep learning algorithm for PDE-based forward and inverse problems". In: *arXiv preprint arXiv:1904.05417*.

Bengio, Yoshua, Yann LeCun, et al. (2007). "Scaling learning algorithms towards AI". In: *Large-scale kernel machines* 34.5, pp. 1–41.

Bhatnagar, Saakaar et al. (2019). "Prediction of aerodynamic flow fields using convolutional neural networks". In: *Computational Mechanics*, pp. 1–21.

Bhattacharya, Kaushik, Nikola B. Kovachki, and Andrew M. Stuart (2020). "Model Reduction and Neural Networks for Parametric PDE(s)". In: *preprint*.

Chi, Lu, Borui Jiang, and Yadong Mu (2020). "Fast Fourier Convolution". In: *Advances in Neural Information Processing Systems* 33.

Cotter, S. L. et al. (Aug. 2013). "MCMC Methods for Functions: Modifying Old Algorithms to Make Them Faster". In: *Statistical Science* 28.3, pp. 424–446. ISSN: 0883-4237. DOI: 10.1214/13-sts421. URL: http://dx.doi.org/10.1214/13-STS421.

DeVore, Ronald A. (2014). "Chapter 3: The Theoretical Foundation of Reduced Basis Methods". In: *Model Reduction and Approximation*. DOI: 10.1137/1.9781611974829.ch3. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611974829.ch3. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611974829.ch3.

E, Weinan and Bing Yu (Mar. 2018). "The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems". English (US). In: *Communications in Mathematics and Statistics*. ISSN: 2194-6701. DOI: 10.1007/s40304-018-0127-z.

Fan, Yuwei, Cindy Orozco Bohorquez, and Lexing Ying (2019). "BCR-Net: A neural network based on the nonstandard wavelet form". In: *Journal of Computational Physics* 384, pp. 1–15.

Fan, Yuwei, Lin Lin, et al. (2019). "A multiscale neural network based on hierarchical matrices". In: *Multiscale Modeling & Simulation* 17.4, pp. 1189–1213.

Greenfeld, Daniel et al. (2019). "Learning to optimize multigrid PDE solvers". In: *International Conference on Machine Learning*. PMLR, pp. 2415–2423.

Guo, Xiaoxiao, Wei Li, and Francesco Iorio (2016). "Convolutional neural networks for steady flow approximation". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hornik, Kurt, Maxwell Stinchcombe, Halbert White, et al. (1989). "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5, pp. 359–366.

Jiang, Chiyu Max et al. (2020). "MeshfreeFlowNet: A Physics-Constrained Deep Continuous Space-Time Super-Resolution Framework". In: *arXiv preprint arXiv:2005.01463*.

Kashinath, Karthik, Philip Marcus, et al. (2020). "Enforcing Physical Constraints in CNNs through Differentiable PDE Layer". In: *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.

Khoo, Yuehaw, Jianfeng Lu, and Lexing Ying (2017). "Solving parametric PDE problems with artificial neural networks". In: *arXiv preprint arXiv:1707.03351*.

Kochkov, Dmitrii et al. (2021). "Machine learning accelerated computational fluid dynamics". In: *arXiv preprint arXiv:2102.01010*.

Li, Zongyi et al. (2020a). *Multipole Graph Neural Operator for Parametric Partial Differential Equations*. arXiv: 2006.09535 [cs.LG].

– (2020b). "Neural operator: Graph kernel network for partial differential equations". In: *arXiv preprint arXiv:2003.03485*.

Lu, Lu, Pengzhan Jin, and George Em Karniadakis (2019). "DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators". In: *arXiv preprint arXiv:1910.03193*.

Mathieu, Michael, Mikael Henaff, and Yann LeCun (2013). *Fast Training of Convolutional Networks through FFTs*. arXiv: 1312.5851 [cs.CV].

Nelsen, NH and AM Stuart (2020). "The Random Feature Model for Input-Output Maps between Banach Spaces". In: *arXiv preprint arXiv:2005.10224*.

Pan, Shaowu and Karthik Duraisamy (2020). "Physics-informed probabilistic learning of linear embeddings of nonlinear dynamics with guaranteed stability". In: *SIAM Journal on Applied Dynamical Systems* 19.1, pp. 480–509.

Patel, Ravi G et al. (2021). "A physics-informed operator regression framework for extracting data-driven continuum models". In: *Computer Methods in Applied Mechanics and Engineering* 373, p. 113500.

Raissi, Maziar, Paris Perdikaris, and George E Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378, pp. 686–707.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241.

Sitzmann, Vincent et al. (2020). "Implicit neural representations with periodic activation functions". In: *arXiv preprint arXiv:2006.09661*.

Smith, Jonathan D, Kamyar Azizzadenesheli, and Zachary E Ross (2020). "Eikonet: Solving the eikonal equation with deep neural networks". In: *arXiv preprint arXiv:2004.00361*.

Wang, Rui et al. (2020). "Towards physics-informed deep learning for turbulent flow prediction". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1457–1466.

Zhu, Yinhao and Nicholas Zabaras (2018). "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification". In: *Journal of Computational Physics*. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.04.018. URL: http://www.sciencedirect.com/science/article/pii/S0021999118302341.

Chapter 5

LEARNING: DISSIPATIVE LOSS FOR OPERATOR LEARNING

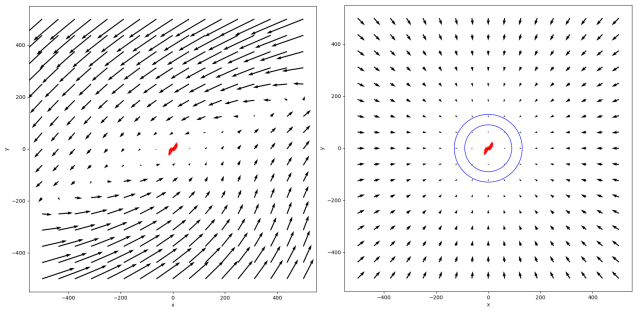
Chaotic systems are notoriously challenging to predict because of their sensitivity to perturbations. Small changes in the initialization or errors during time-stepping accumulate and lead to vastly diverging trajectories. Despite this unpredictable behavior, in dissipative chaotic systems, the long term trajectories converge to a low dimensional set, known as the global attractor. Furthermore this set supports an invariant measure which determines the statistical properties of long-term trajectories. In this work, we propose a machine learning framework for dissipative chaotic systems guided by the goal of accurate statistical prediction. The global attractor, and its invariant measure, are determined by the Markov operator that maps the evolution of the system during infinitesimal time steps. By learning the Markov operator, we can predict the statistical behavior of dissipative chaotic systems, even if we cannot predict exact trajectories. We learn the operator with only the local one-step evolution information and then compose the learned operator to obtain the global attractor and its invariant measure. We further develop a data augmentation technique to impose dissipativity on the learned model, enforcing the desirable property that the composition does not lead to blow ups. Experiments show improved prediction of statistics associated with chaotic dynamical systems, in comparison with previous methods; our experiments are conducted on the Lorenz-63 system, the Kuramoto-Sivashinsky equation and the Navier-Stokes equation with Reynolds number up to 5000.

5.1 Introduction

Machine learning methods for chaotic systems. Chaotic systems are characterized by strong instabilities. A tiny perturbation in the initial condition leads to a completely different trajectory. Such instability makes chaotic systems challenging, both for mathematical analysis and numerical simulation. Because of its intrinsic instability, it is infeasible for any method to capture the exact trajectory of a chaotic system for long periods. Therefore, prior works either fit RNNs on extremely short trajectories or only learn a step-wise projection from a randomly generated evolution using reservoir computing (RC) (Vlachas, Byeon, et al., 2018; Pathak et al., 2018; Vlachas, Pathak, et al., 2020; Fan et al., 2020). These previous attempts are able to

push the limits of faithful prediction to moderate periods on low dimension systems such as ordinary differential equations (ODEs), e.g. the Lorenz '63 system or one-dimensional partial differential equations (PDEs), e.g. the Kuramoto-Sivashinsky (KS) equation. However, they are less effective at modeling more complicated turbulent systems such as the Navier-Stokes equation (NS). Indeed, predicting long trajectories of such chaotic systems is an ill-posed problem and we cannot expect such attempts to be successful. Instead, we take a new perspective: we aim to capture statistical properties of long trajectories, even if we cannot precisely predict them.

Invariants in chaos. Despite the instability, many chaotic systems exhibit certain reproducible statistical properties, such as the auto-correlation and, for PDEs, the energy spectrum. Such properties remain the same in different realizations of the initial condition (Gleick, 2011). This is provably the case for the Lorenz '63 model Tucker, 1999; Holland and Melbourne, 2007 and empirically holds for numerous dissipative PDEs, such as the KS equation and the two-dimensional NS equation (Kolmogorov flows) (Temam, 2012). Mathematically, this behavior is characterized by the existence of invariant measures. Furthermore there exists an **attractor** which is a set towards which the system tends to evolve. For dissipative systems there is a compact set which any given trajectory will enter in a finite time, depending on the initial condition, and thereafter remain inside. The attractor is defined by mapping all initial conditions from this compact set, forward in time (technically it is the $\omega-$limit set of the compact set A. Stuart and A.R. Humphries, 1998). While learning infinite time-horizon trajectories is intractable, it is possible to approximate the attractor, and statistics of trajectories on it, using the Markov operator. The dissipativity property helps to make this problem tractable (A. Stuart and A.R. Humphries, 1998; Humphries and Stuart, 1994). For Markovian systems, samples from this measure can be obtained through access to a **Markov operator**, a memoryless deterministic operator that captures the evolution of the dynamics along an infinitesimal time step. In this case, the Markov operator forms a discrete semigroup, defined by the compositions of Markov operators (A. Stuart and A.R. Humphries, 1998). By learning the Markov operator, we are able to generate the attractor and estimate the invariant measure for a variety of chaotic systems that are of interest to the physics and applied mathematics communities (Fukami, Fukagata, and Taira, 2021; Cardesa, Vela-Martín, and Jiménez, 2017; Chandler and Kerswell, 2013; Koltai and Weiss, 2020; Bramburger, Kutz, and Brunton, 2021; Page, Brenner, and Kerswell,



The flow map extrapolated on a larger domain without (left) and with (right) the enforced dissipativity. The red points are training points on the attractor. The dissipativity is imposed by augmenting the data on the blue shell.

Figure 5.1: Enforcing dissipativity on the Lorenz 63 system.

2021).

Neural operators. To learn the Markov operators for PDEs, we need to model the time-evolution of functions in infinite-dimensional function spaces. This is especially challenging when we need to generate long trajectories. Since even a small error will accumulate over multiple compositions of the learned operator, potentially causing exponential build up or collapse. Because we study the evolution of functions time, we propose to use the recently developed operator learning method known as the neural operator (Li et al., 2020b; Kovachki et al., n.d.). The neural operator remedies the mesh-dependent nature of finite-dimensional operator methods such as RNNs, CNNs, and RC. Neural operators are guaranteed to universally approximate any operator in a mesh independent manner, and hence, can capture the Markov operator of chaotic systems. This local guarantee plus the absorption of trajectories by the global attractor makes it possible to accurately follow the it over long time horizons, allowing us to capture the invariant measure of chaotic systems.

Our contributions. In this work, we formulate a machine learning framework for chaotic systems exploiting their ergodicity and Markovian property. We learn a Markov neural operator (MNO) given only one-step evolution data and compose it over a long horizons to obtain the global attractor of the chaotic system, as shown in Figure 5.2. The MNO forms a discrete semigroup defined by the composition of operators, allowing us to reach any state further in the future (Sviridyuk, 1994). The goal is to predict the global attractor which does not collapse or blow up over a long or infinite time horizon, and shares the same distribution and statistics as the true function space trajectories.

- We formulate a machine learning framework to learn the Markov neural operator with only local time evolution training data. At test time, we compose it over a long horizon to obtain the global attractor.
- We study the ability to learn statistical invariant measures, since predicting exact trajectories is generally intractable.
- We impose dissipativity by augmenting the data on an outer shell to enforce that the dynamic evolution stays close to the attractor. The resulting system is stable against large perturbations.
- We study the choice of time steps $\mathrm{d}t$ used for training the MNO and investigate various Sobolev losses in operator learning. Experiments show the Sobolev norm significantly outperforms standard loss functions in preserving invariant statistics such as the spectrum.

Invariant statistics. We study the invariant statistics proposed in the literature such as the Fourier spectrum, the proper orthogonal decomposition (POD), the point-wise distribution, the auto-correlation, and other domain-specific statistics such as turbulence kinetic energy and dissipation rate. These invariant statistics are usually a combination of different orders of derivatives and moments. By using a standard mean square error (MSE) for training, the models often fail to capture the higher frequency information induced from the derivatives. Therefore, we use the Sobolev norm to address higher-order derivatives and moments (Czarnecki et al., 2017; Beatson et al., 2020). This is similar in spirit to pre-multiplying the spectrum by the wavenumber, an approach commonly used in fluid analysis (Smits, McKeon, and Marusic, 2011). When used for training, the Sobolev norm yields a significant improvement in capturing high frequency details compared to the standard MSE or L^2 loss, and therefore we obtain an approximation of the Markov operator that is able to better characterize the invariant measure.

5.2 Problem Setting

We consider potentially infinitely dimensional dynamical systems where the phase space $\mathcal{U}$ is a Banach space and, in particular, a function space on a Lipschitz domain $D \subset \mathbb{R}^d$ (for finite dimensional systems, $\mathcal{U}$ will be a Euclidian space). We

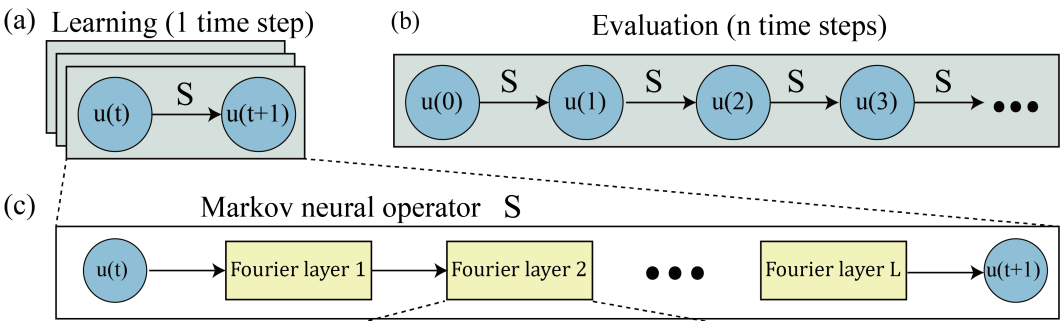


Figure 5.2: Markov neural operator (MNO): learn global dynamics from local data. (a) Learn the MNO from the local time-evolution data. (b) Compose MNO to evaluate the long-time dynamic. (c) Architecture of the MNO.

are interested in the initial-value problem

$$\begin{aligned} \frac{du}{dt}(t) &= F(u(t)), \qquad t \in (0, \infty), \\ u(0) &= u_0, \end{aligned} \tag{5.1}$$

for initial conditions $u_0 \in \mathcal{U}$ where F is usually a non-linear operator. We will assume, given some appropriate boundary conditions on ∂D when applicable, that the solution $u(t) \in \mathcal{U}$ exists and is unique for all times $t \in (0, \infty)$. When making the spatial dependence explicit, if it is present, we will write $u(x, t)$ to indicate the evaluation $u(t)|_x$ for any $x \in D$. We adopt the viewpoint of casting time-dependent PDEs into function space ODEs (5.1), as this leads to the semigroup approach to evolutionary PDEs which underlies our learning methodology.

Dissipativity. Systems for which there exists some bounded, positively-invariant set E such that for any bounded $B \subset \mathcal{U}$, there is some time $t^* = t^*(E, B)$ beyond which the dynamics of any trajectory starting in B enters and remains in E are known as **dissipative systems** (A. Stuart and A.R. Humphries, 1998). The set E is known as the **absorbing set** of the system. For such systems, the global attractor A, defined subsequently, is characterized as the ω-limit set of E. In particular, for any initial condition $u_0 \in \mathcal{U}$, the trajectory $u(t)$ approaches A as $t \to \infty$.

In this work, we consider dissipative dynamical systems where there exist some $\alpha \geq 0$ and $\beta > 0$ such that

$$\frac{1}{2}\frac{d}{dt}||u||^2 \leq \alpha - \beta||u||^2 \tag{5.2}$$

for all $u \in \mathcal{U}$. It can be shown that systems which satisfy this inequality are

dissipative (A. Stuart and A.R. Humphries, 1998) with the absorbing set

$$E = B\left(0, \sqrt{\frac{\alpha}{\beta} + \varepsilon}\right), \tag{5.3}$$

for any $\varepsilon > 0$, where $B(u, r)$ is the open ball centered at $u \in \mathcal{U}$ with radius r.

There are several well-known examples of dynamical systems that satisfy the above inequality. In this paper we consider the finite-dimensional Lorenz-63 system and the infinite-dimensional cases of the Kuramoto-Sivashinsky and 2D incompressible Navier-Stokes equations (Temam, 2012).

Global Attractors. The long time behavior of the solution to (5.1) is characterized by the set $U = U(u_0) \subset \mathcal{U}$ which is **invariant** under the dynamic i.e. $S_t(U) = U$ for all $t \geq 0$, and the orbit $u(t)$ converges

$$\inf_{v \in U} \|u(t) - v\|_{\mathcal{U}} \to 0 \qquad \text{as} \qquad t \to \infty.$$

When it exists, U is often identified as the ω-limit set of u_0. The chaotic nature of certain dynamical systems arises due to the complex structure of this set because $u(t)$ follows U and U can be, for example, a fractal set. A compact, invariant set A is called a **global attractor** if, for any bounded set $B \subset \mathcal{U}$ and any $\epsilon > 0$ there exists a time $t^* = t^*(\epsilon, A, B)$ such that $S_t(B)$ is contained within an ϵ-neighborhood of A for all $t \geq t^*$. Many PDEs arising in physics such as reaction-diffusion equations describing chemical dynamics or the Navier-Stokes equation describing the flow of fluids are dissipative and possess a global attractor which is often times finite-dimensional (Temam, 2012). Therefore, numerically characterizing the attractor is an important problem in scientific computing with many potential applications.

Data distribution. For many applications, an exact form for the possible initial conditions to (5.1) is not available; it is therefore convenient to use a stochastic model to describe the initial states. To that end, let μ_0 be a probability measure on $\mathcal{U}$ and assume that all possible initial conditions to (5.1) come as samples from μ_0 i.e. $u_0 \sim \mu_0$. Then any possible state of the dynamic (5.1) after some time $t > 0$ can be thought of as being distributed according to the pushforward measure $\mu_t \coloneqq S_t^\sharp \mu_0$ i.e. $u(t) \sim \mu_t$. Therefore as the dynamic evolves, so does the type of likely functions that result. This further complicates the problem of long time predictions since training data may only be obtained up to finite time horizons hence the model will need the ability to predict not only on data that is out-of-sample but also out-of-distribution.

Ergodic systems. To alleviate some of the previously presented challenges, we consider **ergodic** systems. Roughly speaking, a system is ergodic if there exists an **invariant measure** μ such that after some time $t^* > 0$, we have $\mu_t \approx \mu$ for any $t \geq t^*$ (in fact, μ can be defined without any reference to μ_0 or its pushforwards, see (Pavliotis and Andrew Stuart, 2008) for details). That is, after some large enough time, the distribution of possible states that the system can be in is fixed for any time further into the future. Indeed, μ charges the global attractor A. Notice that ergodicity is a much more general property than having **stationary states** which means that the system has a fixed period in time, or having **steady states** which means the system is unchanged in time.

Ergodicity mitigates learning a model that is able to predict out-of-distribution since both the input and the output of $\hat{S}_h$ will approximately be distributed according to μ. Furthermore, we may use $\hat{S}_h$ to learn about μ since sampling it simply corresponds to running the dynamic forward. Indeed, we need only generate data on a finite time horizon in order to learn $\hat{S}_h$, and, once learned, we may use it to sample μ indefinitely by simply repeatedly composing $\hat{S}_h$ with itself. Having samples of μ then allows us to compute statistics which characterize the long term behavior of the system and therefore the global attractor A. Note further that this strategy avoids the issue of accumulating errors in long term trajectory predictions since we are only interested in the property that $\hat{S}_h(u(t)) \sim \mu$.

Notably, the existence of a global attractor does not imply the existence of an invariant measure. Indeed, the only deterministic and chaotic systems that are proven to possess an invariant measure are certain ODEs such as the Lorenz-63 system (Holland and Melbourne, 2007). On the other hand, proving the existence of an invariant measure for deterministic and chaotic PDEs such as the KS or NS equations are still open problems, despite ergodic behavior being observed empirically.

5.3 Learning the Markov neural operator in Chaotic Dynamics

We propose the Markov neural operator, a method for learning the underlying Markov operators of chaotic dynamical systems. In particular, we approximate the operator mapping the solution from the current step to the next step $\hat{S}_h : u(t) \mapsto u(t+h)$. The architecture of the Markov neural operator is detailed in Figure 5.2.

Markov operators and the semigroup

Since the system (5.1) is autonomous, that is, F does not explicitly depend on time, under the well-posedness assumption, we may define, for any $t \in [0, \infty)$, a Markov operator $S_t : \mathcal{U} \to \mathcal{U}$ such that $u(t) = S_t u(0)$. This map satisfies the properties

1. $S_0 = I$,
2. $S_t(u_0) = u(t)$,
3. $S_t(S_s(u_0)) = u(t + s)$,

for any $s, t \in [0, \infty)$ and any $u_0 \in \mathcal{U}$ where I denotes the identity operator on $\mathcal{U}$. In particular, the family $\{S_t : t \in [0, \infty)\}$ defines a semigroup of operators acting on $\mathcal{U}$. Our goal is to approximate a particular element of this semigroup associated to some fixed time step $h > 0$ given observations of the trajectory from (5.1). We build an approximation $\hat{S}_h : \mathcal{U} \to \mathcal{U}$ such that

$$\hat{S}_h \approx S_h. \tag{5.4}$$

Long-term predictions. Note that having access to the map $\hat{S}_h$ from (5.4) allows for approximating long time trajectories of (5.1) by repeatedly composing $\hat{S}_h$ with its own output. Indeed, by (5.4) and the semigroup property, we can approximate the evolution by composing the local Markov operators

$$u(n \cdot h) \approx \hat{S}_h^n(u_0) \coloneqq \underbrace{(\hat{S}_h \circ \cdots \circ \hat{S}_h)}_{n \text{ times}}(u_0) \tag{5.5}$$

for any $n \in \mathcal{N}$. This fact is formalized in Theorem 5.3.1 which is stated subsequently.

Invariant statistics. A useful application of the Markov operators is to estimate statistics of the invariant measure of a chaotic system. Assume the target system is ergodic and there exists an invariant measure μ such that $u(t) \sim \mu$ for any t as discussed in Section 5.2. An invariant statistic is defined as

$$T_G \coloneqq \int_{\mathcal{U}} \mathcal{G}(u) \, \mathrm{d}\mu(u) = \lim_{T \to \infty} \frac{1}{T} \int_0^T \mathcal{G}(u(t)) \, \mathrm{d}t \tag{5.6}$$

for any functional $\mathcal{G} : \mathcal{U} \to \mathbb{R}^d$. Examples include the L^2 norm, any spectral coefficients, and the spatial correlation, as well as problem-specific statistics such as the turbulence kinetic energy and dissipation rates in fluid flow problems. Given

the property (5.5) and using the ergodicity from (5.6), the approximate model $\hat{S}_h$ can be used to estimate any invariant statistic simply by computing

$$T_G \approx \frac{1}{n} \sum_{k=1}^{n} G(\hat{S}_h^k u_0)$$

for some $n = T/h$ and $T > 0$ large enough. Examples of fast approximation $\hat{S}_h$ which accurately predict invariant statics are given in Section 8.5.

Enforcing dissipativity. In practice, training data for dynamical systems is typically drawn from trajectories that lie close to the global attractor of the system, so a priori there is no guarantee of a learned model's behavior far from the attractor. Thus, if we seek to learn the global attractor and invariant statistics of a dynamical system, it is crucial that we place contraints on the model to enforce that it learns a dissipative dynamical system.

In particular, given some Markov operator mapping between time-steps $\hat{S}_h : u(t) \mapsto u(t+h)$ and cost functional $C_D : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$, we supplement the loss function with the additional term

$$\mathbb{E}_{u\sim\nu}\left[C_D\left(\hat{S}_h(u), u\right)\right] = \int_{\mathcal{U}} \left|\hat{S}_h(u) - \lambda u\right|^2 \, \mathrm{d}\nu(u), \tag{5.7}$$

up to some scaling constant with respect to the other terms in the loss function, where $0 < \lambda < 1$ is some constant factor for scaling down (i.e., enforcing dissipativity) inputs u drawn from probability measure ν. We choose ν to be a uniform probability distribution supported on some shell with a fixed inner and outer radii from the origin in $\mathcal{U}$. Our choice of cost functional C_D as given in eq. 5.7 scales down u by some constant factor λ, but in principle alternative dissipative cost functionals can be used. The norm $|\cdot|$ used in our C_D is task-specific (e.g., Euclidean norm in the case of the Lorenz-63 system).

We find that enforcing this dissipativity constraint on a shell at a sufficiently large radius encourages the learned Markov operator to produce dissipative predictions arbitrarily far away from the shell. In Section 8.5 we demonstrate that the constraint in eq. 5.7 prevents blow-up of a Markov operator trained on the turbulent Kolmogorov flow system.

Backbone model for $\hat{S}_h$

The above semigroup formulation can be applied with various choices of the backbone model $\hat{S}_h$. In general, we prefer models that can be evaluated quickly and have

approximation guarantees so the per-step error can be controlled. Therefore, we choose the standard feed-forward neural network Park and Sandberg, 1991 for ODE systems, and the Fourier neural operator Li et al., 2020a for infinite dimensional PDE systems.

For the the neural operator parametric class, we prove the following theorem regarding the Markov neural operator. The result states that our construction can approximate trajectories of infinite-dimensional dynamical systems arbitrary well. The proof is given in the Appendix.

Theorem 5.3.1 *Let $K \subset \mathcal{U}$ be a compact set and assume that, for some $h > 0$, the Markov operator $S_h : \mathcal{U} \to \mathcal{U}$ associated to the dynamic* (5.1) *is locally Lipschitz. Then, for any $n \in \mathcal{N}$ and $\epsilon > 0$ there exists a neural operator $\hat{S}_h : \mathcal{U} \to \mathcal{U}$ such that*

$$\sup_{u_0 \in K} \sup_{k \in \{1,\dots,n\}} \|u(kh) - \hat{S}_h^k(u_0)\|_{\mathcal{U}} < \epsilon.$$

Theorem 5.3.1 indicates that of choice of backbone model is rich enough to approximate many chaotic dynamical systems for a arbitrarily long period. For finite-dimensional systems, the same theorem holds with feed-forward neural networks instead of neural operators. We note that standard neural network such as RNNs and CNNs do not possess such approximation theorems in the infinite-dimensional setting. Furthermore we expect that it is possible to show that the global attractor can be approximated arbitrarily well by adapting ideas from the standard theory of numerical integrators such as (A. R. Humphries and A. M. Stuart, 1994).

5.4 Experiments

In this section, we motivate our proposed approach for enforcing dissipativity described in Section 5.3 by applying it to learn a dissipative neural network model for the Lorenz-63 system. We also present the experiment for the Kuramoto-Sivashinsky equation and the Kolmogorov flow raising from the 2-d Navier-Stokes equation. Both PDEs exhibit chaotic behavior.

Lorenz-63 system

To motivate and justify our framework for learning chaotic systems in the infinite-dimensional setting (e.g., Navier-Stokes equations), we first apply our framework (e.g., enforcing model dissipativity and evaluating invariant statistics) on the relatively simple Lorenz-63 ODE system.

Lorenz-63 (Lorenz, 1963) is a system of coupled ordinary differential equations, and it is typically considered to be the simplest dynamical system that exhibits chaotic properties. The Lorenz-63 system is given by

$$\begin{aligned} \dot{u}_x &= \alpha(u_y - u_x), \\ \dot{u}_y &= -\alpha u_x - u_y - u_x u_z, \\ \dot{u}_z &= u_x u_y - b u_z - b(r + \alpha) \end{aligned} \tag{5.8}$$

In this paper, we use the canonical parameters of $(\alpha, b, r) = (10, 8/3, 28)$.

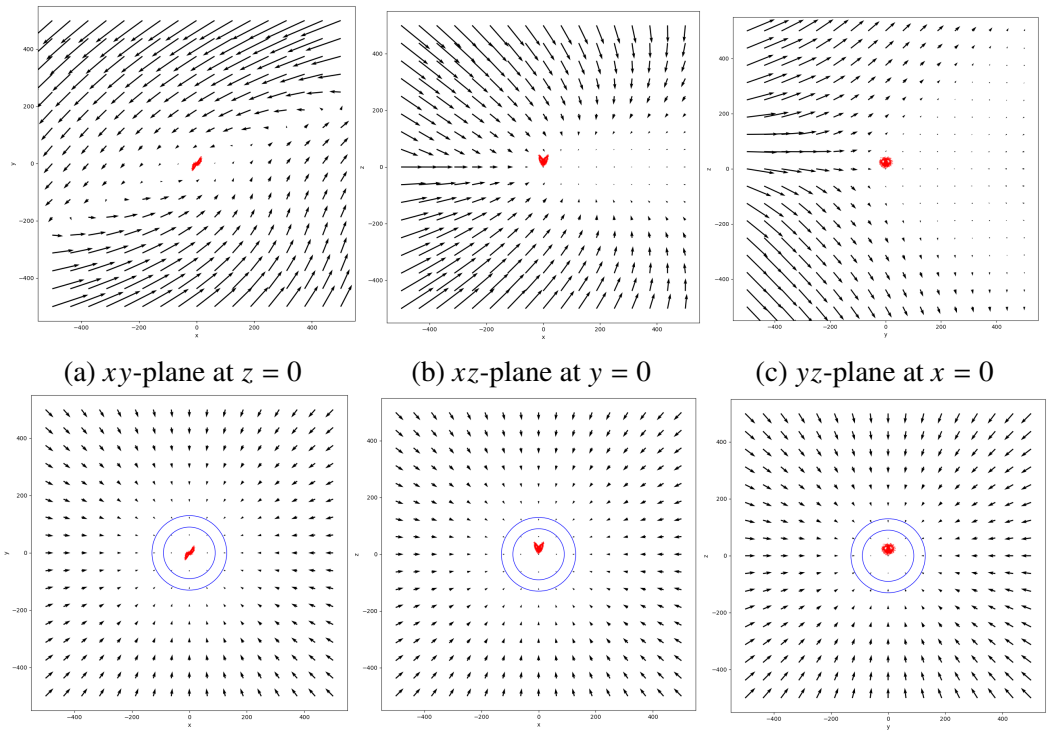
Learning the Markov operator. Since the Markov operator of the Lorenz-63 system is finite-dimensional, we learn the Markov operator by training a simple feedforward neural network on a single trajectory on the Lorenz attractor. Specifically, we use a neural network with 6 hidden layers and 150 neurons per layer. We discretize the training trajectory into time-steps of 0.05 seconds. We train two networks: one with dissipativity enforced and one without.

Enforcing dissipativity. We enforce dissipativity during training with the criterion described in eq. 5.7, with $\lambda = 0.5$ and ν being a uniform probability distribution supported on a shell around the origin with inner radius 90 and outer radius 130. As shown in figure 5.1, enforcing dissipativity produces predictions that isotropically point towards the attractor, implying that the attractive properties of the Lorenz attractor are learned in the process. Observe that the dissipative network is also dissipative outside the shell in which dissipativity was enforced during training. We believe this property of ReLU networks prevents model blow-up even in more difficult learning problems .

Results. We find that enforcing dissipativity does not reduce the relative L2 error compared to the baseline neural network. Further, the invariant statistics of the dissipative model aligns well with the ground-truth data distribution. Our results suggest that dissipativity can be enforced without significantly affecting the model's step-wise error and the learned statistsical properties of the attractor.

In the setting of learning Markovian operator, we consider the following one-dimensional Kuramoto-Sivashinsky equation,

$$\begin{aligned} \frac{\partial u}{\partial t} &= -u\frac{\partial u}{\partial x} - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^4 u}{\partial x^4}, && \text{on } [0, L] \times (0, \infty) \\ u(\cdot, 0) &= u_0, && \text{on } [0, L] \end{aligned} \tag{5.9}$$



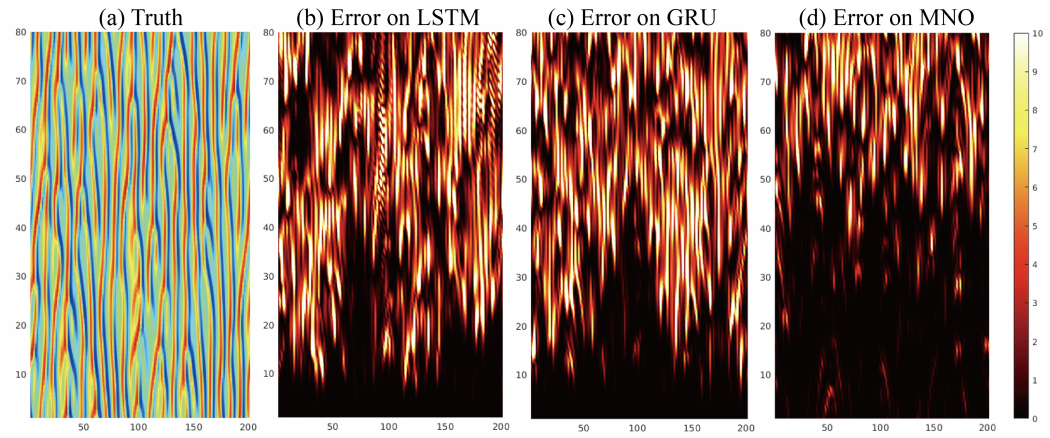
(a) xy-plane at $z = 0$ (b) xz-plane at $y = 0$ (c) yz-plane at $x = 0$

(d) xy-plane at $z = 0$ (e) xz-plane at $y = 0$ (f) yz-plane at $x = 0$

Figure 5.3: Predicted flow field of dissipative network on Lorenz-63. The red points are training points on the attractor. Dissipativity is enforced uniformly within the blue shell.

where $L = 32\pi$ or 64π and the spatial domain $[0, L]$ is equipped with periodic boundary conditions. We assume the initial condition $u_0 \in \dot{L}^2_{\text{per}}([0, L]; \mathbb{R})$, where $\dot{L}^2_{\text{per}}([0, L]; \mathbb{R})$ is the space of all mean zero L^2-functions that are periodic on $[0, L]$. Existence of the semigroup $S_t : \dot{L}^2_{\text{per}}([0, L]; \mathbb{R}) \to \dot{L}^2_{\text{per}}([0, L]; \mathbb{R})$ is established in Temam, 2012, Theorem 3.1. Data is obtained by solving the equation using the exponential time-differencing fourth-order Runge-Kutta method from (Kassam and Trefethen, 2005). Random initial conditions are generated according to a mean zero Gaussian measure with covariance $L^{-2/\alpha}\tau^{\frac{1}{2}(2\alpha-1)}(-\Delta + (\tau^2/L^2)I)^{-\alpha}$ where $\alpha = 2$, $\tau = 7$, and periodic boundary conditions on $[0, L]$; for details see (Lord, Powell, and Shardlow, 2014).

Invariant statistics for the KS equations As shown in Figure 5.5, we present enormous invariant statistics for the KS equation. We use 150000 snapshots to train the MNO, LSTM, and GRU to model the evolution operator of the KS equation with $h = 1s$. We compose each model for $T = 1000$ time steps to obtain a long trajectory (attractor), and estimate various invariant statistics from them.



The x-axis is the spatial domain; the y-axis is the temporal domain. The figure shows that LSTM and GRU start to diverge at $t = 20s$ while MNO is able to keep up with the exact trajectory until $t = 50s$.

Figure 5.4: Trajectory and error on the KS equation

Choice of time discretization. We further study the choice of time steps h. As shown in Figure 5.6, when the time steps are too large, the correlation is chaotic and hard to capture. But counter-intuitively, when the time steps are too small, the evolution is also hard to capture. In this case, the input and output of the network will be very close, and the identity map will be a local minimum. An easy fix is to use MNO to learn the time-derivative or residual. This is shown in the figure, where the residual model (blue line) has a better per-step error and accumulated error at smaller h. When the time step is large, there is no difference in modeling the residual. This idea can generalize to other integrators as an extension of Neural ODEs to PDEs (Chen et al., 2018).

Navier-Stokes equation

We also consider the two-dimensional Navier-Stokes equation for a viscous, incompressible fluid,

$$\begin{aligned} \frac{\partial u}{\partial t} &= -u \cdot \nabla u - \nabla p + \frac{1}{Re}\Delta u + \sin(ny)\hat{x}, && \text{on } [0, 2\pi]^2 \times (0, \infty) \\ \nabla \cdot u &= 0 && \text{on } [0, 2\pi]^2 \times [0, \infty) \\ u(\cdot, 0) &= u_0 && \text{on } [0, 2\pi]^2 \end{aligned} \tag{5.10}$$

where u denotes the velocity, p the pressure, and $Re > 0$ is the Reynolds number. The domain $[0, 2\pi]^2$ is equipped with periodic boundary conditions. The specific choice of forcing $\sin(ny)\hat{x}$ constitutes a Kolmogorov flow; we choose $n = 4$ in all experiments. We define $\mathcal{U}$ to be the closed subspace of $L^2([0, 2\pi]^2; \mathbb{R}^2)$,

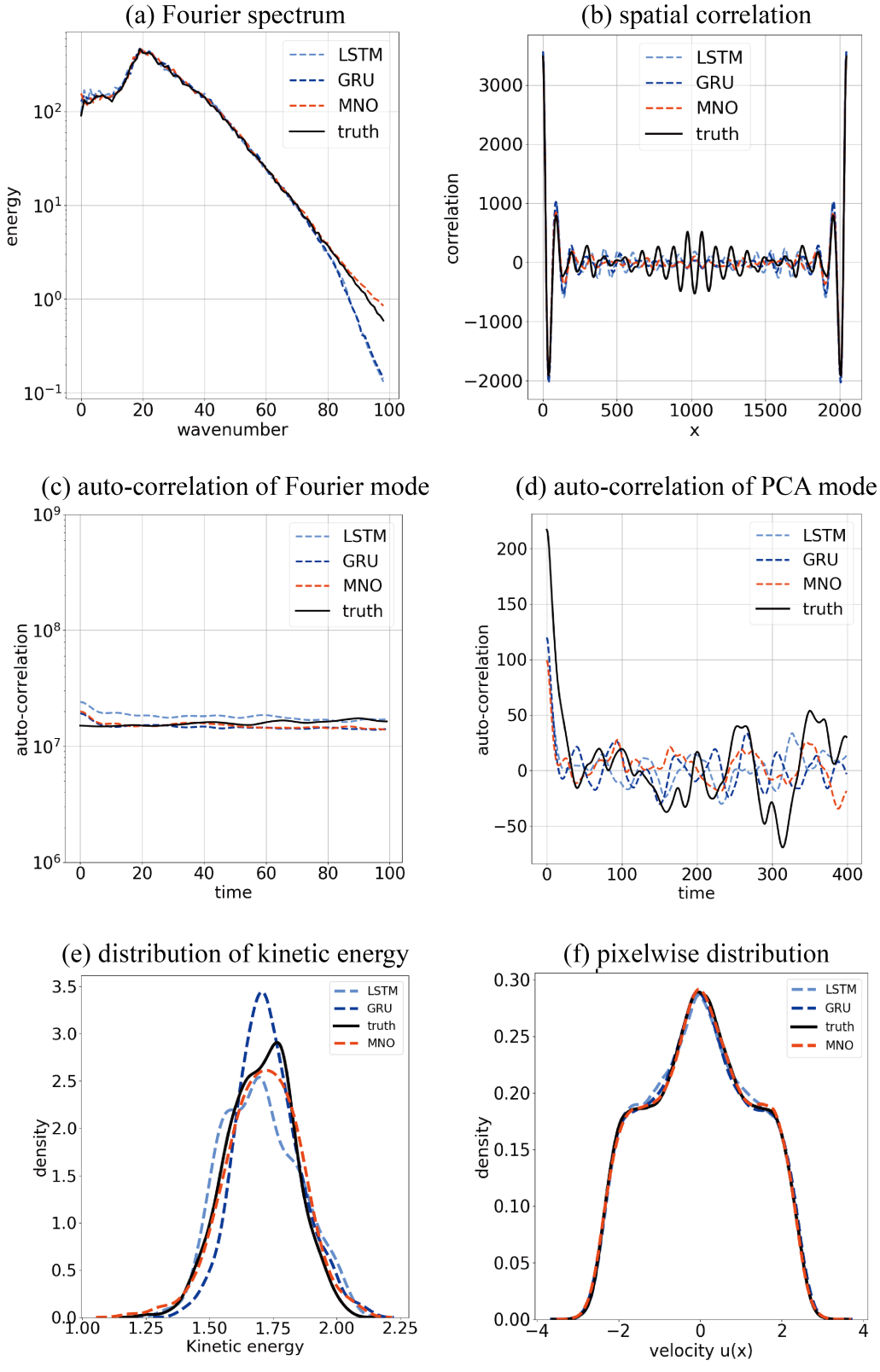


Figure 5.5: Invariant statistics for the KS equation

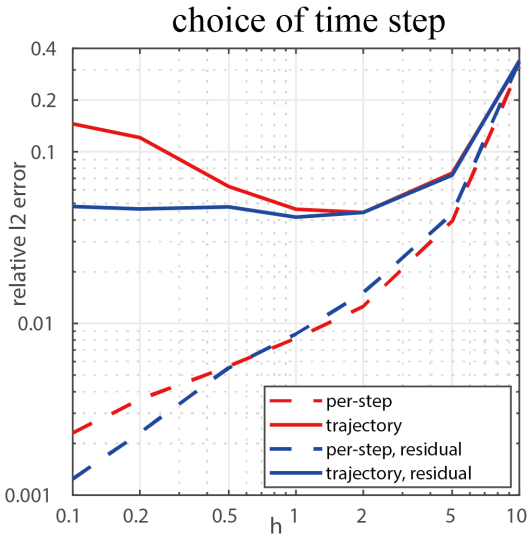
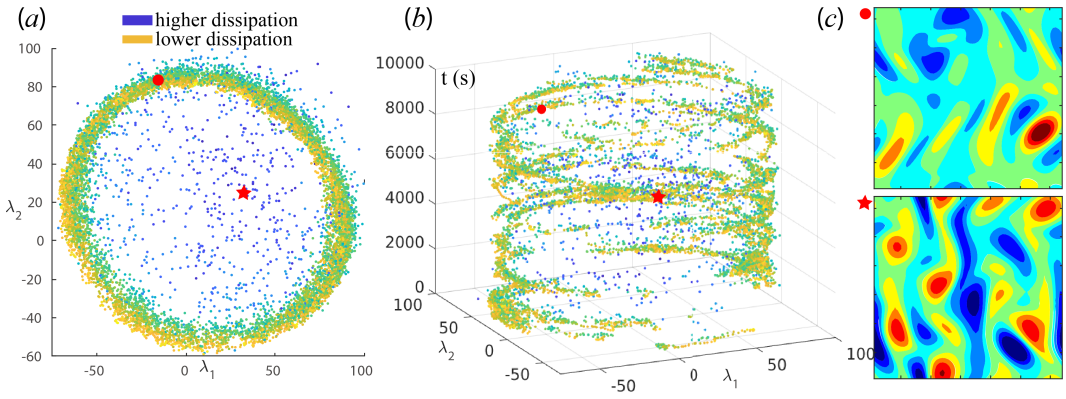


Figure 5.6: Choice of time step

$\mathcal{U} = \{u \in \dot{L}^2_{\text{per}}([0, 2\pi]^2; \mathbb{R}^2) : \nabla \cdot u = 0\}$ and assume $u_0 \in \mathcal{U}$. We define the vorticity $w = (\nabla \times u)\hat{z}$ and the stream function f as the solution to the Poisson equation $-\Delta f = w$. Existence of the semigroup $S_t : \mathcal{U} \to \mathcal{U}$ is established in Temam, 2012, Theorem 2.1. We denote turbulence kinetic energy (TKE) $\langle (u - \bar{u})^2 \rangle$, and dissipation $\epsilon = \langle w^2 \rangle / Re$. Data is obtained by solving the equation in vorticity form using the pseudo-spectral split step method from (Chandler and Kerswell, 2013). Random initial conditions are generated according to a mean zero Gaussian measure with covariance $7^{3/2}(-\Delta + 49I)^{-2.5}$ with periodic boundary conditions on $[0, 2\pi]^2$.

Benchmarks for 2d Navier-Stokes. We compare MNO with common standard two-dimensional dynamic models including U-Net(Ronneberger, Fischer, and Brox, 2015) and LSTM-CNN(Shi et al., 2015) on modeling the vorticity w. We choose the discretization $h = 1s$. The training dataset consists of 180 realizations of trajectories on time interval $t \in [100, 500]$ (the first 100 seconds are discarded) which adds up to $180 \times 400 = 72,000$ snapshots in total. Another 20 realizations are generated for testing. Each single snapshot has resolution 64×64. We use the Adam optimizer to minimize the relative L^2 loss with learning rate = 0.0005, and step learning rate scheduler that decays by half every 10 epochs for 50 epochs in total. **U-Net:** we use five layers of convolution and deconvolution with width from 64 to 1024. **LSTM-CNN:** we use one layer of LSTM with width = 64. **MNO:** we parameterize the 2-d Fourier neural operator consists of four Fourier layers with 20 frequencies



The 10000 time steps trajectory generated by MNO projected onto the first two components of PCA. Each point corresponds to an snapshot on the attractor. Two points are selected for further visualization of vorticity field.

Figure 5.7: The learned attractor of the Kolmogorov flow.

Model	training	loss	$\mathbf{L^2}$ error	$\mathbf{H^1}$ error	$\mathbf{H^2}$ error	TKE error	$\boldsymbol{\epsilon}$ error
MNO	L^2 loss	0.0166	0.0187	0.0474	0.1729	**0.0136**	0.0303
	H^1 loss	0.0184	0.0151	0.0264	0.0656	0.0256	**0.0017**
	H^2 loss	0.0202	**0.0143**	**0.0206**	**0.0361**	0.0226	0.0193
U-Net	L^2 loss	0.0269	0.0549	0.1023	0.3055	0.0958	0.0934
	H^1 loss	0.0377	0.0570	0.0901	0.2164	0.1688	0.1127
	H^2 loss	0.0565	0.0676	0.0936	0.1749	0.0482	0.0841
ConvLSTM	L^2 loss	0.2436	0.2537	0.3854	1.0130	0.0140	24.1119
	H^1 loss	0.2855	0.2577	0.3308	0.5336	0.6977	6.9167
	H^2 loss	0.3367	0.2727	0.3311	0.4352	0.8594	4.0976

Table 5.1: Benchmark on vorticity for the Kolmogorov flow with Re= 40

per channel and width = 64.

Enforcing dissipativity. We enforce dissipativity during training with the criterion described in eq. 5.7, with $\lambda = 0.5$ and ν being a uniform probability distribution supported on a shell around the origin.

Accuracy with respect to various norms. MNO shows near one order of magnitude better accuracy compared to U-Net and LSTM-CNN. As shown in Table 5.1, we train each model using the balanced $L^2(= H^0)$, H^1, and H^2 losses, defined as the sum of the relative L^2 loss grouped by each order of derivative. And we measure the error with respect to the standard (unbalanced) norms. The MNO with H^2 loss consistently achieves the smallest error on vorticity on all of the L^2, H^1, and

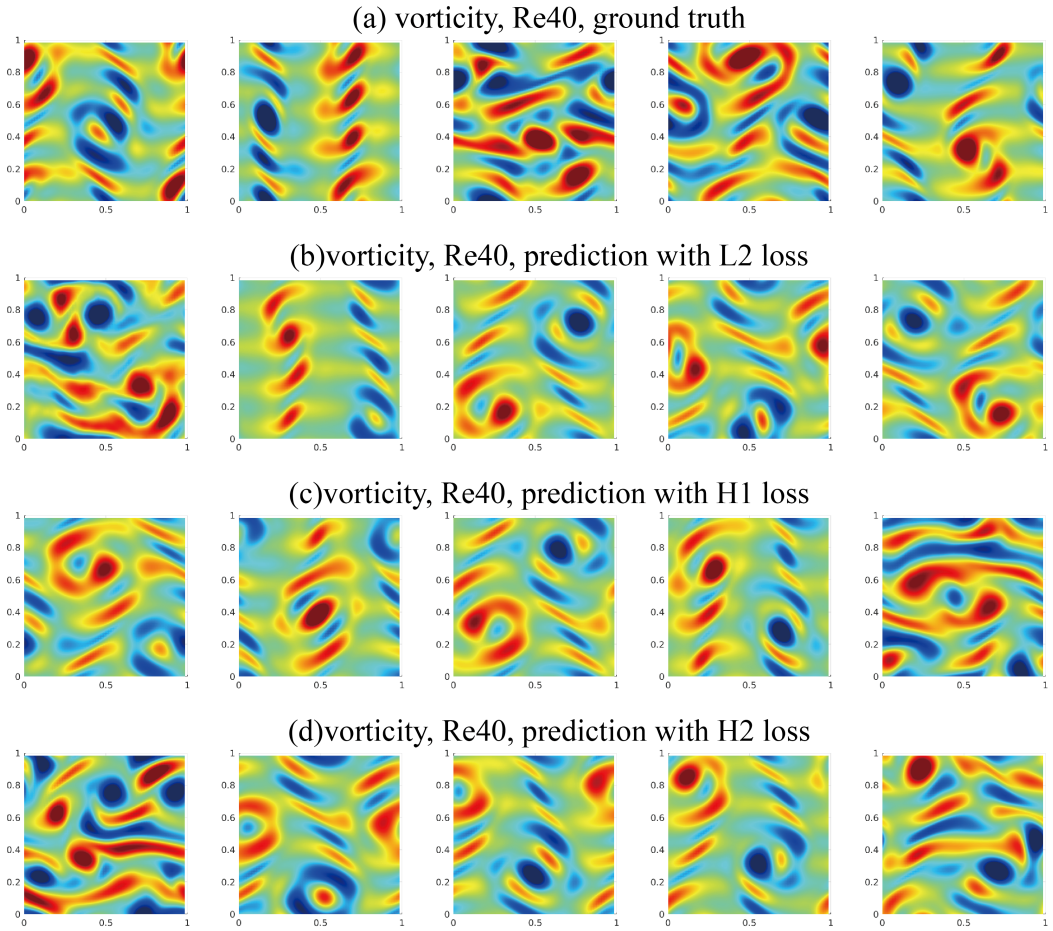


Figure 5.8: Visualization of the NS equation, Re40

H^2 norms. However, L^2 loss achieves the smallest error on the turbulence kinetic energy (TKE); H^1 loss achieves the smallest error on the dissipation ϵ.

The NS equation with Re40 is shown in Figure 5.8. (a) show the ground truth data of vorticity field, each column represents a snapshot at $t = 100s$ with a different initial condition. (b), (c), (d) show the predicted trajectory of MNO on vorticity, using L^2, H^1, and H^2 losses respectively. We are able to generate a long trajectory with the MNO model. The five columns represent $t = 1000s, 2000s, 3000s, 4000s, 5000s$ respectively. As shown in the figure, the predicted trajectories (b) (c) (d) share the same behaviors as in the ground truth (a). It indicates the MNO model is stable.

Estimate the attractor and invariant statistics. We compose MNO 10000 times to obtain the global attractor, and we compute the PCA (POD) basis of these 10000 snapshots and project them onto the first two components. As shown in Figure (a), we obtain a cycle-shaped attractor. The true attractor has a degree of freedom

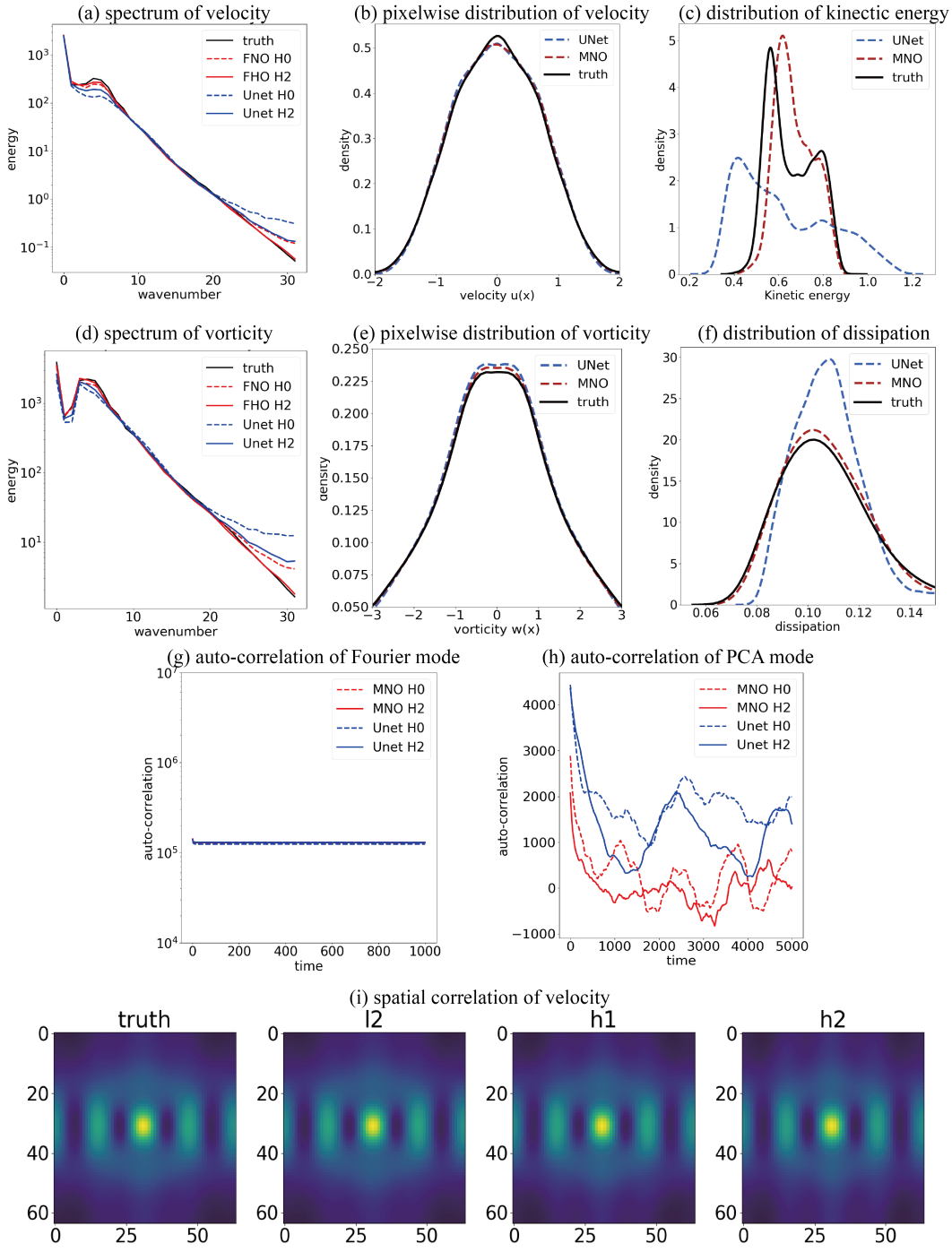


Figure 5.9: Invariant statistics for the NS equation

around $O(100)$ (Temam, 2012). If the attractor is a high-dimensional sphere, then most of the mass concentrates around its equator. Therefore, when projected to low-dimension, the attractor will have the shape of a ring. Most of the points are located on the ring, while a few other points are located in the center. The points in the center have high dissipation, implying they are intermittent states. In Figure (b) we add the time axis. While the trajectory jumps around the cycle, we observe there is a rough period of 2000s. We perform the same PCA analysis on the training data, which shows the same behavior. Similarly, we present enormous invariant statistics for the NS equation (Re40), as shown in Figure 5.9,. We use 72000 snapshots to train the MNO, UNet, and ConvLSTM to model the evolution operator of the KS equation with $h = 1s$. We compose each model for $T = 10000$ time steps to obtain a long trajectory (attractor), and estimate various invariant statistics from them.

Order of derivatives. Roughly speaking, vorticity is the derivative of velocity; velocity is the derivative of the stream function. Therefore we can denote the order of derivative of vorticity, velocity, and stream function as 2, 1, and 0 respectively. Combining vorticity, velocity, and stream function, with L^2, H^1, and H^2 loss, we have in total the order of derivatives ranging from 0 to 4. We observe, in general, it is best practice to keep the order of derivatives in the model at a number slightly higher than that of the target quantity. For example, as shown in Figure 5.10, when querying the velocity (first-order quantity), it is best to use second-order (modeling velocity plus H^1 loss or modeling vorticity plus L^2 loss). This is further illustrated in Table 5.2. In general, using a higher order of derivatives as the loss will increase the power of the model and capture the invariant statistics more accurately. However, a higher-order of derivative means higher irregularity. It in turn requires a higher resolution for the model to resolve and for computing the discrete Fourier transform. This trade-off again suggests it is best to pick a Sobolev norm not too low or too high.

5.5 Discussion and Conclusion

In this work, we propose a machine learning framework that trains from local data and predict the global attractor and invariant statistic of chaotic systems. The Markov operator forms a discrete semigroup defined by the composition of operators that empirically does not collapse or blow up over a long or infinite time horizon. Experiments also show MNO predicts the attractor that shares the same distribution and statistics as the true function space trajectories. The Markov operator and

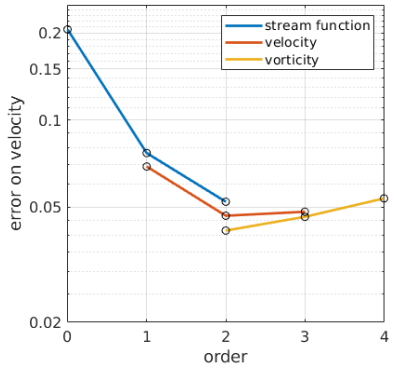


Figure 5.10: Error on velocity with respect to the order of derivative.

Model	training	loss	(order)	error on f	error on u	error on w
f	L^2 loss	0.0379	(0^{th} order)	0.0383	0.2057	2.0154
f	H^1 loss	0.0512	(1^{st} order)	0.0268	0.0769	0.3656
f	H^2 loss	0.0973	(2^{nd} order)	0.0198	0.0522	0.2227
u	L^2 loss	0.0688	(1^{st} order)	0.0217	0.0691	0.3217
u	H^1 loss	0.1246	(2^{nd} order)	**0.0170**	0.0467	0.1972
u	H^2 loss	0.2662	(3^{rd} order)	0.0178	0.0482	0.1852
w	L^2 loss	0.1710	(2^{nd} order)	0.0219	**0.0415**	0.1736
w	H^1 loss	0.3383	(3^{rd} order)	0.0268	0.0463	**0.1694**
w	H^2 loss	0.4590	(4^{th} order)	0.0312	0.0536	0.1854

Table 5.2: Vorticity w, velocity u, and stream function f for the Kolmogorov flow with $Re = 500$

chaotic systems are mathematically interesting and physically relevant. MNO shows potential for application studying the bifurcation of complicated system, for example, in the modeling of chemical reactions and the flow of turbulent fluids.

This work provides a method for fast computation in many scientific computing problems. These methods have two main long-term impacts beyond the immediate interests of scientific computing communities. Since our methods are orders of magnitude faster than traditional solvers that are prominently used in supercomputers, edge devices, and servers, the deployment of our methods significantly reduces the carbon footprints caused by scientific studies. Furthermore, the proposed methods are extremely flexible. The off the shelf usage of our methods allows scientists from a variety of disciplines, from chemistry, biology, ecology, epidemiology, to physics and applied mathematics to deploy our methods to their complex PDE of interest (or any other problem that involves learning maps between function spaces).

MNO has two major limitations. First, it assumes the target system is approximately Markovian. If the system is heavily path-dependent, then the MNO framework does not directly apply. Second, although we develop an approximation theorem for finite period, it does not hold for infinite time horizon.

As discussed previously, it is infeasible to track the exact trajectory of chaotic systems on an infinite time horizon. Even very small errors will accumulate in each step, and eventually cause the simulation to diverge from the true trajectory. However, it is possible to track the attractor of the system. An attractor is absorbing. If the simulated trajectory only makes a small error, the attractor will absorb it back, so that the simulated trajectory will never diverge from the true attractor. Therefore, it is possible to have the simulated trajectory capture the true attractor.

To obtain an infinite-time approximation error bound is non-trivial. Previously, (A. Stuart and A.R. Humphries, 1998; Humphries and Stuart, 1994) (cf. Theorem 3.12) show a result for (finite-dimensional) ODE systems. If the system is Lipschitz then there exists a numerical simulation that forms a dissipative dynamical system that does not blow up or collapse. And the the simulated attractor $\mathcal{A}_h$ approximates the true attractor $\mathcal{A}$ with the time step h

$$dist(\mathcal{A}_h, \mathcal{A}) \to 0, \quad \text{as } h \to 0$$

To generalize such theorem to Markov neural operator (MNO), we need to overcome two difficulties (1) generalize the formulation from (finite-dimensional) ODE systems to (infinite-dimensional) PDE systems, and (2) show MNO can obtain a sufficient error rate with respect to the time step h.

The first aspect requires extending the theory from finite dimension to infinite dimension, which is non-trivial since the operator F in (5.1) is not compact or bounded. This makes it hard to bound the error with respect to the attractor (Andrew Stuart, 1995). The second aspect requires to formulate MNO slightly differently. In the current formulation, the evolution operator is chosen for a fixed time step h. To achieve $O(h)$ error we need to formulate the evolution operator continuously for infinitesimal h. Especially, for a semi-linear PDE system

$$\frac{du}{dt} + Au = F(u)$$

where A is a linear, self-adjoint operator and F is a continuous but nonlinear operator (This formulation includes the KS and NS equations). The evolution can be written

as

$$u(t+h) = e^{-Ah}u(t) + \int_0^h e^{-A(h-s)}F(u(t+s))ds$$

Where $\Phi(u(t), A, t) := \int_0^h e^{-A(h-s)}F(u(s))ds$ is bounded despite F is not. If one can approximate $\Phi(u(t), A, h)$ by a neural operator, then MNO can potentially achieve the needed error rate. This shows hope to obtain an approximation error bound for infinite time zero. We leave this as a promising future direction.

References

Beatson, Alex et al. (2020). "Learning composable energy surrogates for PDE order reduction". In: *Advances in Neural Information Processing Systems* 33.

Bramburger, Jason J, J Nathan Kutz, and Steven L Brunton (2021). "Data-Driven Stabilization of Periodic Orbits". In: *IEEE Access* 9, pp. 43504–43521.

Cardesa, José I, Alberto Vela-Martín, and Javier Jiménez (2017). "The turbulent cascade in five dimensions". In: *Science* 357.6353, pp. 782–784.

Chandler, Gary J and Rich R Kerswell (2013). "Invariant recurrent solutions embedded in a turbulent two-dimensional Kolmogorov flow". In: *Journal of Fluid Mechanics* 722, pp. 554–595.

Chen, Tian Qi et al. (2018). "Neural ordinary differential equations". In: *Advances in neural information processing systems*, pp. 6571–6583.

Czarnecki, Wojciech Marian et al. (2017). "Sobolev training for neural networks". In: *arXiv preprint arXiv:1706.04859*.

Fan, Huawei et al. (2020). "Long-term prediction of chaotic systems with machine learning". In: *Physical Review Research* 2.1, p. 012080.

Fukami, Kai, Koji Fukagata, and Kunihiko Taira (2021). "Machine-learning-based spatio-temporal super resolution reconstruction of turbulent flows". In: *Journal of Fluid Mechanics* 909.

Gleick, James (2011). *Chaos: Making a new science*. Open Road Media.

Holland, Mark and Ian Melbourne (Oct. 2007). "Central limit theorems and invariance principles for Lorenz attractors". In: *Journal of the London Mathematical Society* 76.2, pp. 345–364.

Humphries, A. R. and A. M. Stuart (1994). "Runge–Kutta Methods for Dissipative and Gradient Dynamical Systems". In: *SIAM Journal on Numerical Analysis* 31.5, pp. 1452–1485.

– (1994). "Runge–Kutta methods for dissipative and gradient dynamical systems". In: *SIAM journal on numerical analysis* 31.5, pp. 1452–1485.

Kassam, Aly-Khan and Lloyd N. Trefethen (2005). "Fourth-Order Time-Stepping for Stiff PDEs". In: *SIAM Journal on Scientific Computing* 26.4, pp. 1214–1233.

Koltai, Péter and Stephan Weiss (2020). "Diffusion maps embedding and transition matrix analysis of the large-scale flow structure in turbulent Rayleigh–Bénard convection". In: *Nonlinearity* 33.4, p. 1723.

Kovachki, Nikola B. et al. (n.d.). "Neural Operator: Learning Maps Between Function Spaces". In: *In Preparation* ().

Li, Zongyi et al. (2020a). "Fourier neural operator for parametric partial differential equations". In: *arXiv preprint arXiv:2010.08895*.

– (2020b). "Neural operator: Graph kernel network for partial differential equations". In: *arXiv preprint arXiv:2003.03485*.

Lord, Gabriel J, Catherine E Powell, and Tony Shardlow (2014). *An introduction to computational stochastic PDEs*. Vol. 50. Cambridge University Press.

Lorenz, Edward N (1963). "Deterministic nonperiodic flow". In: *Journal of atmospheric sciences* 20.2, pp. 130–141.

Page, Jacob, Michael P Brenner, and Rich R Kerswell (2021). "Revealing the state space of turbulence using machine learning". In: *Physical Review Fluids* 6.3, p. 034402.

Park, Jooyoung and Irwin W Sandberg (1991). "Universal approximation using radial-basis-function networks". In: *Neural computation* 3.2, pp. 246–257.

Pathak, Jaideep et al. (2018). "Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach". In: *Physical review letters* 120.2, p. 024102.

Pavliotis, Grigorios and Andrew Stuart (Jan. 2008). *Multiscale Methods: Averaging and Homogenization*. Vol. 53. ISBN: 978-0-387-73828-4. DOI: 10.1007/978-0-387-73829-1.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241.

Shi, Xingjian et al. (2015). "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *arXiv preprint arXiv:1506.04214*.

Smits, Alexander J, Beverley J McKeon, and Ivan Marusic (2011). "High–Reynolds number wall turbulence". In: *Annual Review of Fluid Mechanics* 43, pp. 353–375.

Stuart, A. and A.R. Humphries (1998). *Dynamical Systems and Numerical Analysis*. Cambridge Monographs on Applie. Cambridge University Press. ISBN: 9780521645638.

Stuart, Andrew (1995). "Perturbation theory for infinite dimensional dynamical systems". In.

Sviridyuk, Georgy Anatolevich (1994). “On the general theory of operator semigroups”. In: *Russian Mathematical Surveys* 49.4, pp. 45–74.

Temam, Roger (2012). *Infinite-dimensional dynamical systems in mechanics and physics*. Vol. 68. Springer Science & Business Media.

Tucker, Warwick (1999). “The Lorenz attractor exists”. In: *Comptes Rendus de l’Academie des Sciences - Series I - Mathematics* 328.12, pp. 1197–1202. ISSN: 0764-4442.

Vlachas, Pantelis R, Wonmin Byeon, et al. (2018). “Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474.2213, p. 20170844.

Vlachas, Pantelis R, Jaideep Pathak, et al. (2020). “Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics”. In: *Neural Networks*.

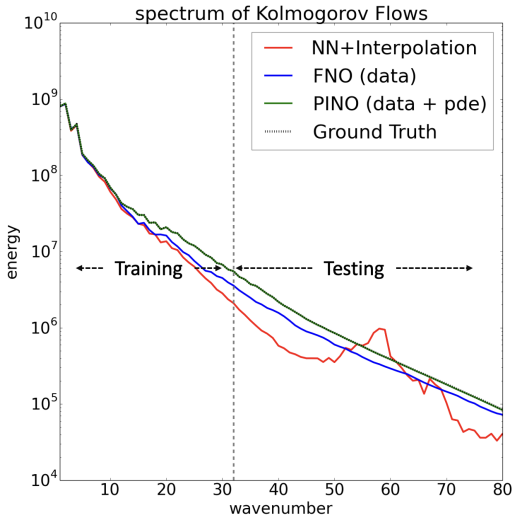
Chapter 6

LEARNING: PHYSICS INFORMED LEARNING FOR NEURAL OPERATOR

In this paper, we propose physics-informed neural operators (PINO) that combine training data and physics constraints to learn the solution operator of a given family of parametric Partial Differential Equations (PDE). PINO is the first hybrid approach incorporating data and PDE constraints at different resolutions to learn the operator. Specifically, in PINO, we combine coarse-resolution training data with PDE constraints imposed at a higher resolution. The resulting PINO model can accurately approximate the ground-truth solution operator for many popular PDE families and shows no degradation in accuracy even under zero-shot super-resolution, i.e., being able to predict beyond the resolution of training data. PINO uses the Fourier neural operator (FNO) framework that is guaranteed to be a universal approximator for any continuous operator and discretization convergent in the limit of mesh refinement. By adding PDE constraints to FNO at a higher resolution, we obtain a high-fidelity reconstruction of the ground-truth operator. Moreover, PINO succeeds in settings where no training data is available and only PDE constraints are imposed, while previous approaches, such as the Physics-Informed Neural Network (PINN), fail due to optimization challenges, e.g., in multi-scale dynamic systems such as Kolmogorov flows.

6.1 Introduction

Machine learning methods have recently shown promise in solving partial differential equations (PDEs) (Kovachki et al., n.d.; Z. Li, Kovachki, et al., 2021a; Z. Li, Kovachki, et al., 2021b; L. Lu, P. Jin, and George Em Karniadakis, 2021; Brunton, Noack, and Koumoutsakos, 2020). A recent breakthrough is the paradigm of operator learning for solving PDEs. Unlike standard neural networks that learn using inputs and outputs of fixed dimensions, neural operators learn operators, which are mappings between function spaces (Kovachki et al., n.d.; Z. Li, Kovachki, et al., 2021a; Z. Li, Kovachki, et al., 2021b). The class of neural operators is guaranteed to be a universal approximator for any continuous operator (Kovachki et al., n.d.) and hence, has the capacity to approximate any operator including any solution operator of a given family of parametric PDEs. Note that the solution operator is



PINO uses both training data and PDE loss function and perfectly extrapolates to unseen frequencies in Kolmogorov Flows. FNO uses only training data and does not have further information on higher frequencies, but still follows the general trend of the ground-truth spectrum. On the other hand, using a trained UNet with trilinear interpolation (NN+Interpolation) has severe distortions at higher frequencies.

Figure 6.1: Physics-Informed Neural Operator (PINO) extrapolates to unseen higher frequencies.

the mapping from the input function (initial and boundary conditions) to the output solution function. Previous works show that neural operators can capture complex multi-scale dynamic processes and are significantly faster than numerical solvers (B. Liu et al., 2022; H. Yang et al., 2022; Pathak, Subramanian, et al., 2023; Z. Li, D. Z. Huang, et al., n.d.; Wen et al., 2023b; Bonev et al., 2023; Wen et al., 2023a).

Neural operators are proven to be discretization convergent in the limit of mesh refinement (Kovachki et al., n.d.), meaning they converge to a continuum operator in the limit as the discretization is refined. Consequently, they can be evaluated at any data discretization or resolution at inference time without the need for retraining. For example, neural operators such as Fourier neural operator (FNO) can extrapolate to frequencies that are not seen during training in Kolmogorov Flows, as shown in Figure 6.1, while standard approaches such as training a UNet and adding trilinear interpolation leads to significantly worse results at higher resolutions.

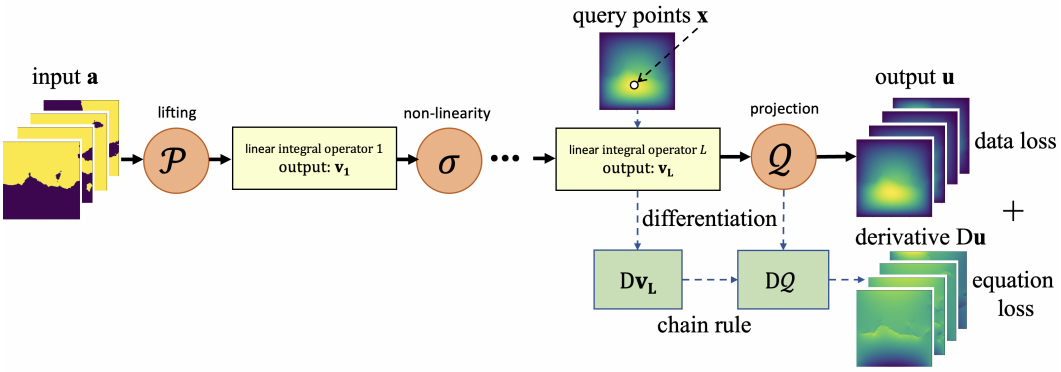
Even though FNO follows the general trend of the Kolmogorov flow in Figure 6.1, it cannot perfectly match it in the regime of super-resolution, i.e., beyond the frequencies seen during training. More generally, neural operators cannot perfectly approximate the ground-truth operator when only coarse-resolution training data is

provided. This is a fundamental limitation of data-driven operator learning methods which depend on the availability of training data, which can come either from existing numerical solvers or direct observations of the physical phenomena. In many scenarios, such data can be expensive to generate, unavailable, or available only as low-resolution observations (Hersbach et al., 2020). This limits the ability of neural operators to learn high-fidelity models. Moreover, the generalization of the learned neural operators to unseen scenarios and conditions that are different from training data is challenging.

Our Approach and Contributions

In this paper, we remedy the above shortcomings of data-driven operator learning methods through the framework of physics-informed neural operators (PINO). Here, we take a hybrid approach of combining training data (when available) with a PDE loss function at a higher resolution. This allows us to approximate the solution operator of many PDE families nearly perfectly. While there have been many physics-informed approaches proposed previously (discussed in 6.1), ours is the first to incorporate PDE constraints at a higher resolution as a remedy for low resolution training data. We show that this results in high-fidelity solution operator approximations. As shown in Figure 6.1, PINO extrapolates to unseen frequencies in Kolmogorov Flows. Thus, we show that the PINO model learned from such multi-resolution hybrid loss functions has almost no degradation in accuracy even on high-resolution test instances when only low-resolution training data is available. Further, our PINO approach also overcomes the optimization challenges in approaches such as Physics-Informed Neural Network (PINN) (Raissi, Perdikaris, and George E Karniadakis, 2019) that are purely based on PDE loss functions and do not use training data, and thus, PINO can solve more challenging problems such as time-dependent PDEs.

PINO utilizes both the data and equation loss functions (whichever are available) for operator learning. To further improve accuracy at test time, we fine-tune the learned operator on the given PDE instance using only the equation loss function. This allows us to provide a nearly-zero error for the given PDE instance at all resolutions. A schematic of PINO is shown in Figure 8.1, where the neural operator architecture is based on Fourier neural operator (FNO) (Z. Li, Kovachki, et al., 2021a). The derivatives needed for the equation loss in PINO are computed explicitly through the operator layers in function spaces. In particular, we efficiently compute the explicit gradients on function space through Fourier-space computations. In



PINO trains neural operator with both training data and PDE loss function. The figure shows the neural operator architecture with the lifting point-wise operator that receives input function a and outputs function v_0 with a larger co-dimension. This operation is followed by L blocks that compute linear integral operators followed by non-linearity, and the last layer of which outputs the function v_L. The pointwise projection operator projects v_L to output function u. Both v_L and u are functions and all their derivatives ($\mathrm{D}v_\mathrm{L}$, $\mathrm{D}u$) can be computed in their exact forms at any query points x.

Figure 6.2: Architecture of Physics-Informed Neural Operator

contrast, previous auto-differentiation methods must compute the derivatives at sampling locations.

The PDE loss function added to PINO vastly improves **generalization** and physical validity in operator learning compared to purely data-driven methods. PINO requires fewer to no training data and generalizes better compared to the data-driven FNO (Z. Li, Kovachki, et al., n.d.), especially on high-resolution test instances. On average, the relative error is 7% lower on both transient and Kolmogorov flows, while matching the speedup of data-trained FNO architecture (400x) compared to the GPU-based pseudo-spectral solver (He and W. Sun, 2007). Further, the PINO model on the Navier-Stokes equation can be easily transferred to different Reynolds numbers ranging from 100 to 500 using instance-wise fine-tuning.

We also use PINO for solving **inverse problems** by either: (1) learning the forward solution operator and using gradient-based optimization to obtain the inverse solution, or (2) learning the inverse solution operator directly. Imposing the PDE loss guarantees the inverse solution is physically valid in both approaches. We find that of these two approaches, the latter is more accurate for recovering the coefficient function in Darcy flow. We show this approach is 3000x faster than the conventional solvers using accelerated Markov Chain Monte-Carlo (MCMC) (Cotter et al., 2013).

Related Work

Learning solution to PDEs has been proposed under two paradigms: (i) data-driven learning and (ii) physics-informed optimization. The former utilizes data from existing solvers or experiments, while the latter is purely based on PDE constraints. An example of data-driven methods is the class of neural operators for learning the solution operator of a given family of parametric PDEs. An example of the physics-based approach is the Physics-Informed Neural Network (PINN) for optimizing the PDE constraints to obtain the solution function of a given PDE instance. Both these approaches have shortcomings. Neural operators require data, and when that is limited or not available, they are unable to learn the solution operator faithfully. PINN, on the other hand, does not require data but is prone to failure, especially on multi-scale dynamic systems due to optimization challenges. Previous work by (Yingzhou Li, J. Lu, and A. Mao, 2020) has shown promise in learning discretized solution map with variational loss. In this work we generalize it to operator learning.

Neural operators learn the solution operator of a family of PDEs, defined by the map from the input–initial conditions and boundary conditions, to the output–solution functions. In this case, usually, a dataset of input-output pairs from an existing solver or real-world observation is given. There are two main aspects to consider (a) models: the design of models for learning highly complicated PDE solution operators, and (b) data: minimizing data requirements and improving generalization. Recent advances in operator learning replace traditional convolutional neural networks and U-Nets from computer vision with operator-based models tailored to PDEs with greatly improved model expressiveness (Z. Li, Kovachki, et al., n.d.; L. Lu, P. Jin, and George Em Karniadakis, 2021; Patel et al., 2021; Rui Wang et al., 2020; Duvall, Duraisamy, and Pan, 2021). Specifically, the neural operator generalizes the neural network to the operator setting where the input and output spaces are infinite-dimensional. The framework has successfully approximated solution operators for highly non-linear problems such as turbulence flow (Z. Li, Kovachki, et al., 2021b; Z. Li, Kovachki, et al., 2021a). However, the data challenges remain. In particular, (1) training data from an existing solver or an experimental setup is costly to obtain, (2) models struggle in generalizing away from the training distribution, and (3) constructing the most efficient approximation for given data remains elusive. Moreover, it is also evident that in many real-world applications, observational data often is available at only low resolutions (Hersbach et al., 2020), limiting the model's ability to generalize.

Alternatives to data-driven approaches for solving PDEs are physics-based approaches that require no training data. A popular framework known as Physics-Informed Neural Network (PINN) (Raissi, Perdikaris, and George E Karniadakis, 2019) uses optimization to find the solution function of a given PDE instance. PINN uses a neural network as the ansatz of the solution function and optimizes a loss function to minimize the violation of the given equation by taking advantage of auto-differentiation to compute the exact derivatives. PINN overcomes the need to choose a discretization grid that most numerical solvers require, e.g., finite difference methods (FDM) and finite element methods (FEM). It has shown promise in solving PDEs for a wide range of applications, including higher dimensional problems. (L. Lu, Meng, Z. Mao, et al., 2021; J. Han, Jentzen, and Weinan, 2018; Hennigh et al., 2021; Kashinath et al., 2021). Recently, researchers have developed many variations of PINN with promising results for solving inverse problems and partially observed tasks (D. Lu et al., 2021; Zhu et al., 2019; Smith et al., 2021).

However, PINN fails in many multi-scale dynamic PDE systems (S. Wang, X. Yu, and Perdikaris, 2022; Fuks and Tchelepi, 2020; Raissi, Yazdani, and George Em Karniadakis, 2020) due to two main reasons, viz., (1) the challenging optimization landscape of the PDE constraints (S. Wang, Teng, and Perdikaris, 2021) and its sensitivity to hyper-parameter selection (L. Sun et al., 2020), and (2) the difficulty in propagating information from the initial or boundary conditions to unseen parts of the interior or to future times (Dwivedi and Srinivasan, 2020). Moreover, PINN only learns the solution function of a single PDE instance and cannot generalize to other instances without re-optimization. Previous work on physics-informed DeepONet that imposes PDE losses on DeepONet (S. Wang, H. Wang, and Perdikaris, 2021) overcomes this limitation and can learn across multiple PDE instances. While the PDE loss is computed at any query points, the input is limited to a fixed grid in standard DeepONet (Kovachki et al., n.d.), and its architecture is a linear method of approximation (Lanthaler et al., 2023). Our work overcomes all previously mentioned limitations. Further, a unique feature that PINO enjoys over other hybrid learning methods (Zhu et al., 2019; Zhang et al., 2022; X. Huang et al., 2022) is its ability to incorporate data and PDE loss functions at different resolutions. This has not been attempted before, and none of the previous works focus on extrapolation to higher resolutions, beyond what is seen in training data.

6.2 Problem Settings

In this section, we first define the stationary and dynamic PDE systems that we consider. We give an overview of the physics-informed setting and operator-learning setting. In the end, we define the Fourier neural operator as a specific model for operator learning.

Problem settings

We consider two natural classes of PDEs. In the first, we consider the stationary system

$$\begin{aligned} \mathcal{P}(u, a) &= 0, && \text{in } D \subset \mathbb{R}^d \\ u &= g, && \text{in } \partial D \end{aligned} \tag{6.1}$$

where D is a bounded domain, $a \in \mathcal{A} \subseteq \mathcal{V}$ is a PDE coefficient/parameter, $u \in \mathcal{U}$ is the unknown, and $\mathcal{P} : \mathcal{U} \times \mathcal{A} \to \mathcal{F}$ is a possibly non-linear partial differential operator with $(\mathcal{U}, \mathcal{V}, \mathcal{F})$ a triplet of Banach spaces. Usually, the function g is a fixed boundary condition but can also potentially enter as a parameter. This formulation gives rise to the solution operator $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ defined by $a \mapsto u$. A prototypical example is the second-order elliptic equation $\mathcal{P}(u, a) = -\nabla \cdot (a \nabla u) + f$.

In the second setting, we consider the dynamical system

$$\begin{aligned} \frac{du}{dt} &= \mathcal{R}(u), && \text{in } D \times (0, \infty) \\ u &= g, && \text{in } \partial D \times (0, \infty) \\ u &= a && \text{in } \bar{D} \times \{0\} \end{aligned} \tag{6.2}$$

where $a = u(0) \in \mathcal{A} \subseteq \mathcal{V}$ is the initial condition, $u(t) \in \mathcal{U}$ for $t > 0$ is the unknown, and $\mathcal{R}$ is a possibly non-linear partial differential operator with $\mathcal{U}$, and $\mathcal{V}$ Banach spaces. As before, we take g to be a known boundary condition. We assume that u exists and is bounded for all time and for every $u_0 \in \mathcal{U}$. This formulation gives rise to the solution operator $\mathcal{G}^\dagger : \mathcal{A} \to C\big((0, T]; \mathcal{U}\big)$ defined by $a \mapsto u$. Prototypical examples include the Burgers' equation and the Navier-Stokes equation.

Solving equation using the physics-informed neural networks

Given an instance a and a solution operator $\mathcal{G}^\dagger$ defined by equations (6.1) or (9.1) , we denote by $u^\dagger = \mathcal{G}^\dagger(a)$ the unique ground truth. The equation-solving task is to approximate $u^\dagger$. This setting consists of the ML-enhanced conventional solvers such as learned finite element, finite difference, and multigrid solvers (Kochkov et al., 2021; Pathak, Mustafa Mustafa, et al., 2021; Greenfeld et al., 2019), as well as

purely neural network-based solvers such as the Physics-Informed Neural Networks (PINNs), Deep Galerkin Method, and Deep Ritz Method (Raissi, Perdikaris, and George E Karniadakis, 2019; Sirignano and Spiliopoulos, 2018; Weinan and B. Yu, 2018). Especially, these PINN-type methods use a neural network u_θ with parameters θ as the ansatz to approximate the solution function $u^\dagger$. The parameters θ are found by minimizing the physics-informed loss with exact derivatives computed using automatic differentiation (autograd). In the stationary case, the physics-informed loss is defined by minimizing the l.h.s. of equation (6.1) in the squared norm of $\mathcal{F}$. A typical choice is $\mathcal{F} = L^2(D)$, giving the loss function

$$\begin{aligned}\mathcal{L}_{\text{pde}}(a, u_\theta) &= \Big\|\mathcal{P}(a, u_\theta)\Big\|^2_{L^2(D)} + \alpha\Big\|u_\theta|_{\partial D} - g\Big\|^2_{L^2(\partial D)} \\ &= \int_D |\mathcal{P}(u_\theta(x), a(x))|^2 \mathrm{d}x + \alpha \int_{\partial D} |u_\theta(x) - g(x)|^2 \mathrm{d}x\end{aligned} \tag{6.3}$$

In the case of a dynamical system, one minimizes the residual of equation (9.1) in some natural norm up to a fixed final time $T > 0$. A typical choice is the $L^2\big((0,T]; L^2(D)\big)$ norm, yielding

$$\begin{aligned}\mathcal{L}_{\text{pde}}(a, u_\theta) &= \Big\|\frac{du_\theta}{dt} - \mathcal{R}(u_\theta)\Big\|^2_{L^2(T;D)} + \alpha\Big\|u_\theta|_{\partial D} - g\Big\|^2_{L^2(T;\partial D)} + \beta\Big\|u_\theta|_{t=0} - a\Big\|^2_{L^2(D)} \\ &= \int_0^T \int_D |\frac{du_\theta}{dt}(t,x) - \mathcal{R}(u_\theta)(t,x)|^2 \mathrm{d}x\mathrm{d}t \\ &\quad + \alpha \int_0^T \int_{\partial D} |u_\theta(t,x) - g(t,x)|^2 \mathrm{d}x\mathrm{d}t \\ &\quad + \beta \int_D |u_\theta(0,x) - a(x)|^2 \mathrm{d}x\end{aligned} \tag{6.4}$$

The PDE loss consists of the physics loss in the interior and the data loss on the boundary and initial conditions, with hyper-parameters $\alpha, \beta > 0$. Alternatively, the optimization can be formulated via the variational form (Weinan and B. Yu, 2018).

Challenges of PINN PINNs take advantage of the universal approximability of neural networks, but, in return, suffer from the low-frequency induced bias. Em-

pirically, PINNs often fail to solve challenging PDEs when the solution exhibits high-frequency or multi-scale structure (S. Wang, Teng, and Perdikaris, 2021; S. Wang, X. Yu, and Perdikaris, 2022; Fuks and Tchelepi, 2020; Raissi, Yazdani, and George Em Karniadakis, 2020). Further, as an iterative solver, PINNs have difficulty propagating information from the initial condition or boundary condition to unseen parts of the interior or to future times (Dwivedi and Srinivasan, 2020). For example, in challenging problems such as turbulence, PINNs are only able to solve the PDE on a relatively small domain (X. Jin et al., 2021), or otherwise, require extra observational data which is not always available in practice (Raissi, Yazdani, and George Em Karniadakis, 2020; Cai et al., 2021). In this work, we propose to overcome the challenges posed by optimization by integrating operator learning with PINNs.

Learning the solution operator via neural operator

An alternative setting is to learn the solution operator $\mathcal{G}$. Given a PDE as defined in (6.1) or (9.1) and the corresponding solution operator $\mathcal{G}^\dagger$, one can use a neural operator $\mathcal{G}_\theta$ with parameters θ as a surrogate model to approximate $\mathcal{G}^\dagger$. Usually we assume a dataset $\{a_j, u_j\}_{j=1}^N$ is available, where $\mathcal{G}^\dagger(a_j) = u_j$ and $a_j \sim \mu$ are i.i.d. samples from some distribution μ supported on $\mathcal{A}$. In this case, one can optimize the solution operator by minimizing the empirical data loss on a given data pair

$$\mathcal{L}_{\text{data}}(u, \mathcal{G}_\theta(a)) = \|u - \mathcal{G}_\theta(a)\|_{\mathcal{U}}^2 = \int_D |u(x) - \mathcal{G}_\theta(a)(x)|^2 \mathrm{d}x \tag{6.5}$$

where we assume the setting of (6.1) for simplicity of the exposition. The operator data loss is defined as the average error across all possible inputs

$$\begin{aligned} \mathcal{J}_{\text{data}}(\mathcal{G}_\theta) = \|\mathcal{G}^\dagger - \mathcal{G}_\theta\|_{L_\mu^2(\mathcal{A};\mathcal{U})}^2 &= \mathbb{E}_{a\sim\mu}[\mathcal{L}_{\text{data}}(\mathcal{G}^\dagger(a), \mathcal{G}_\theta(a))] \\ &\approx \frac{1}{N}\sum_{j=1}^N \int_D |u_j(x) - \mathcal{G}_\theta(a_j)(x)|^2 \mathrm{d}x. \end{aligned} \tag{6.6}$$

Similarly, one can define the operator PDE loss as

$$\mathcal{J}_{\text{pde}}(\mathcal{G}_\theta) = \mathbb{E}_{a\sim\mu}[\mathcal{L}_{\text{pde}}(a, \mathcal{G}_\theta(a))]. \tag{6.7}$$

In general, it is non-trivial to compute the derivatives $d\mathcal{G}_\theta(a)/dx$ and $d\mathcal{G}_\theta(a)/dt$ for model $\mathcal{G}_\theta$. In the following section, we will discuss how to compute these derivatives for Fourier neural operator.

Neural operators

In this work, we will focus on the neural operator model designed for the operator learning problem (Z. Li, Kovachki, et al., n.d.). The neural operator is formulated as a generalization of standard deep neural networks to the operator setting. Neural operator composes linear integral operator $\mathcal{K}$ with pointwise non-linear activation function σ to approximate highly non-linear operators.

Definition 6.2.1 (Neural operator $\mathcal{G}_\theta$) *Define the neural operator*

$$\mathcal{G}_\theta := \mathcal{Q} \circ (\mathcal{W}_L + \mathcal{K}_L) \circ \cdots \circ \sigma(\mathcal{W}_1 + \mathcal{K}_1) \circ \mathcal{P} \tag{6.8}$$

where $\mathcal{P}$ and $\mathcal{Q}$ are pointwise operators, parameterized with neural networks $P : \mathbb{R}^{d_a} \to \mathbb{R}^{d_1}$ and $Q : \mathbb{R}^{d_L} \to \mathbb{R}^{d_u}$, where d_a is the co-dimension of an input function $a \in \mathcal{A}$ and d_u is the co-dimension of the output function u. $\mathcal{P}$ operator lifts the lower dimension function into higher dimensional space and $\mathcal{Q}$ operator projects the higher dimension function back to the lower dimensional space. The model stacks L layers of $\sigma(\mathcal{W}_l + \mathcal{K}_l)$ where $\mathcal{W}$ are pointwise linear operators parameterized as matrices $W_l \in \mathbb{R}^{d_{l+1} \times d_l}$, $\mathcal{K}_l : \{D \to \mathbb{R}^{d_l}\} \to \{D \to \mathbb{R}^{d_{l+1}}\}$ are integral kernel operators, and σ are fixed activation functions. The parameters θ consists of all the parameters in $\mathcal{P}, \mathcal{Q}, \mathcal{W}_l, \mathcal{K}_l$.

Definition 6.2.2 (Kernel Integral Operators) *We define the kernel integral operator $\mathcal{K}$ used in* (9.18)*. Let $\kappa^{(l)} \in C(D \times D; \mathbb{R}^{d_{l+1} \times d_l})$ and let ν be a Borel measure on D. Then we define $\mathcal{K}$ by*

$$(\mathcal{K}v_l)(x) = \int_D \kappa^{(l)}(x, y) v_l(y)\, d\nu(y) \qquad \forall x \in D. \tag{6.9}$$

The kernel integral operator can be discretized and implemented with graph neural networks as in graph neural operators (Z. Li, Kovachki, et al., n.d.).

$$(\mathcal{K}v_l)(x) = \sum_{B(x)} \kappa^{(l)}(x, y) v_l(y) \qquad \forall x \in D. \tag{6.10}$$

where $B(x)$ is a ball of center at x. As a generalization, the kernel function can also take the form of $(\mathcal{K}v_l)(x) = \sum_{B(x)} \kappa^{(l)}(x, y, v_l(y))$.

Recently, (Z. Li, Kovachki, et al., 2021a) proposes the Fourier neural operator (FNO) that restricts the integral operator $\mathcal{K}$ to convolution. In this case, it can apply the Fast Fourier Transform (FFT) to efficiently compute $\mathcal{K}$. This leads to a fast architecture that obtains state-of-the-art results for PDE problems.

Definition 6.2.3 (Fourier convolution operator) *One specific form of the kernel integral operator is the Fourier convolution operator*

$$\big(\mathcal{K}v_l\big)(x) = \mathcal{F}^{-1}\Big(R \cdot (\mathcal{F}v_l)\Big)(x) \qquad \forall x \in D \tag{6.11}$$

where $\mathcal{F}, \mathcal{F}^{-1}$ are the Fast Fourier transform and its inverse; R is part of the parameter θ to be learn.

One can build a neural operator with mixed kernel integral layers and Fourier convolution layers. If the input and output query points are sampled from non-uniform mesh, we can use the graph kernel operator as the first and last layer for continuous evaluation, while using the Fourier layers in the middle for efficient computation, similar to (Z. Li, D. Z. Huang, et al., n.d.).

Challenges of operator learning. Operator learning is similar to supervised learning in computer vision and language where data play a very important role. One needs to assume the training points and testing points follow the same problem setting and the same distribution. Especially, the previous FNO model trained on one coefficient (e.g. Reynolds number) or one geometry cannot be easily generalized to another. Moreover, for more challenging PDEs where the solver is very slow or the solver is even not existent, it is hard to gather a representative dataset. On the other hand, since prior training methods for FNO do not use any knowledge of the equation, the trained models cannot get arbitrarily close to the ground truth by using the higher resolution as in conventional solvers, leaving a gap of generalization error. These challenges limit the applications of the prior works beyond accelerating the solver and modeling real-world experiments. In the following section, we will introduce the PINO framework to overcome these problems by using the equation constraints.

Discretization convergent. Resolution and discretization convergence is defined as obtaining a unique continuum operator in the limit of mesh refinement (Kovachki et al., n.d.). The work (Bartolucci et al., 2023) recently introduced a new concept of representation equivalence, which requires zero aliasing error at each layer, which PINO does not fulfill. When all the Fourier modes in FNO are active, an aliasing error is inevitably present. However, in many practical applications, this is typically not an issue, and degraded performance due to aliasing is rarely observed, since the maximum number of modes in an FNO is typically far fewer

than the grid size. In fact, the non-linear transformations allow for re-capturing the truncated high-frequency modes which allows for generalization beyond the see training data. Requiring representation equivalence leads to linear methods of approximation which are know to be sub-optimal in their representation capacity (Lanthaler et al., 2023).

Related work. Many machine learning models have been explored for operator learning (Bhattacharya et al., 2021; Nelsen and Stuart, 2021; Patel et al., 2021). Besides the above line of work, the deep operator network (DeepONet) is one of the most famous operator models that have shown enormous promise in applications (L. Lu, P. Jin, and George Em Karniadakis, 2021). The work from (Kontolati et al., 2023) compares the polynomial chaos expansion (PCE), DeepONet, and FNO, and shows that DeepONet has a higher approximation accuracy over PCE. According to Figure 5 in (Kontolati et al., 2023), standard DeepONet and FNO share a similar convergence rate. A similar comparison study is reported by de Hoop et. al. (Hoop et al., 2023) where FNO seems to converge faster. We choose FNO as our base model for its scalability to large problems.

6.3 Physics-Informed Neural Operator

We propose the Physics-Informed Neural Operator (PINO) framework that uses one neural operator model $\mathcal{G}_\theta$ for solving both operator learning problems and equation solving problems. It consists of two phases

- **Operator learning**: learn a neural operator $\mathcal{G}_\theta$ to approximate the target solution operator $\mathcal{G}^\dagger$ using either/both the data loss $\mathcal{J}_{data}$ or/and the PDE loss $\mathcal{J}_{pde}$.
- **Instance-wise fine-tuning**: use $\mathcal{G}_\theta(a)$ as the ansatz to approximate $u^\dagger$ with the pde loss $\mathcal{L}_{pde}$ and/or an additional operator loss $\mathcal{L}_{op}$ obtained from the operator learning phase.

Physics-informed operator learning

For operator learning, we use the physics constraints $\mathcal{J}_{pde}$ and supervision from data to train the neural operator. Especially one can sample an unlimited amount of virtual PDE instances by drawing additional initial conditions or coefficient conditions $a_j \sim \mu$ for training. In this sense, we have access to the unlimited dataset by sampling new input a_j in each training iteration. This advantage of using

PDE constraints removes the requirement on the dataset and makes the supervised problem into a semi-supervised one.

While PINO can be trained with physics constraints $\mathcal{J}_{pde}$ only, the $\mathcal{J}_{data}$ can provide stronger supervision than physics constraints and thus make the optimization much easier. PINO leverages the supervision from any available data to combine with physics constraints for better optimization landscape and thus learning accurate neural operators. A special case is to train a neural operator on low-resolution data instances with high-resolution PDE constraint, which will be studied in section 6.4.

Instance-wise fine-tuning of trained operator ansatz

Given a learned operator $\mathcal{G}_\theta$, we use $\mathcal{G}_\theta(a)$ as the ansatz to solve for $u^\dagger$. The optimization procedure is similar to PINNs where it computes the PDE loss $\mathcal{L}_{pde}$ on a, except that we propose to use a neural operator instead of a neural network. Since the PDE loss is a soft constraint and challenging to optimize, we also add an optional operator loss $\mathcal{L}_{op}$ (anchor loss) to bound the further fine-tuned model from the learned operator model

$$\mathcal{L}_{op}\big(\mathcal{G}_{\theta_i}(a), \mathcal{G}_{\theta_0}(a)\big) \coloneqq \|\mathcal{G}_{\theta_i}(a) - \mathcal{G}_{\theta_0}(a)\|^2_{\mathcal{U}}$$

where $\mathcal{G}_{\theta_i}(a)$ is the model at i^{th} training epoch. We update the operator $\mathcal{G}_\theta$ using the loss $\mathcal{L}_{pde} + \alpha\mathcal{L}_{op}$. It is possible to further apply optimization techniques to fine-tune the last fewer layers of the neural operator and progressive training that gradually increase the grid resolution and use finer resolution in test time.

Optimization landscape. Using the operator as the ansatz has two major advantages: (1) PINN does point-wise optimization, while PINO does optimization in the space of functions. In the linear integral operation $\mathcal{K}$, the operator parameterizes the solution function as a sum of the basis function. Optimization of the set of coefficients and basis is easier than just optimizing a single function as in PINNs. (2) we can learn these basis functions in the operator learning phase which makes the later instance-wise fine-tuning even easier. In PINO, we do not need to propagate the information from the initial condition and boundary condition to the interior. It just requires fine-tuning the solution function parameterized by the solution operator.

Trade-off. (1) complexity and accuracy: instance-wise fine-tuning is an option to spend more computation in exchange for better accuracy. The learned operator is extremely fast since it is performing inference on the neural operator. On the

other hand, instance-wise fine-tuning can improve accuracy while incurring more computational costs. (2) resolution effects on optimization landscape and truncation error (i.e. the error of the numerical differentiation): using a higher resolution and finer grid will reduce the truncation error. However, it has a higher computational complexity and memory consumption. A higher resolution may also potentially make the optimization unstable. Using hard constraints such as the anchor loss $\mathcal{L}_{op}$ relieves such a problem.

Derivatives of neural operators

In order to use the equation loss $\mathcal{L}_{pde}$, one of the major technical challenges is to efficiently compute the derivatives $D(\mathcal{G}_\theta a) = d(\mathcal{G}_\theta a)/dx$ for neural operators. In this section, we discuss three efficient methods to compute the derivatives of the neural operator $\mathcal{G}_\theta$ as defined in (9.18).

Numerical differentiation. A simple but efficient approach is to use conventional numerical derivatives such as finite difference and Fourier differentiation (Zhu et al., 2019; Gao, L. Sun, and J.-X. Wang, 2021). These numerical differentiation methods are fast and memory-efficient: given a n-points grid, finite difference requires $O(n)$, and the Fourier method requires $O(n \log n)$. These differentiation methods are agnostic to the underlying neural network architecture. It can be applied to the neural operator with Graph layer 6.2.2 or Fourier layer 6.2.3 or neural networks such as UNet.

However, the numerical differentiation methods face the same challenges as the corresponding numerical solvers: finite difference methods require a fine-resolution uniform grid; spectral methods require smoothness and uniform grids. Especially. These numerical errors on the derivatives will be amplified on the output solution.

Pointwise differentiation with autograd. Similar to PINN (Raissi, Perdikaris, and George E Karniadakis, 2019), the most general way to compute the exact derivatives is to use the auto-differentiation library of neural networks (autograd). To apply autograd, one needs to use a neural network to parameterize the solution function $u : x \mapsto u(x)$. However, it is not straightforward to write out the solution function in the neural operator which directly outputs the numerical solution $u = \mathcal{G}_\theta(a)$ on a grid, especially for FNO which uses FFT. To apply autograd, we design a query function u that input x and output $u(x)$. Recall $\mathcal{G}_\theta \coloneqq Q \circ (\mathcal{W}_L + \mathcal{K}_L) \circ \cdots \circ \sigma(\mathcal{W}_1 + \mathcal{K}_1) \circ \mathcal{P}$ and $u = \mathcal{G}_\theta a = Q v_L = Q(\mathcal{W}_L + \mathcal{K}_L) v_{L-1} \ldots$. Since Q is

pointwise,

$$u(x) = \mathcal{Q}(v_L)(x) = Q(v_L(x)) = Q\big((\mathcal{W}_L v_{L-1})(x) + \mathcal{K}_L v_{L-1}(x)\big) \tag{6.12}$$

For both the kernel operator and Fourier operator, we either remove the pointwise residual term of the last layer $(\mathcal{W}_L v_{L-1})(x)$ or define the query function as an interpolation function on $\mathcal{W}_L$.

For kernel integral operator 6.2.2, the kernel function can directly take the query points are input. So the query function

$$u(x) = Q\big(\sum_{B(x)} \kappa^{(l)}(x, y, v_{L-1}(y))\big)$$

where we omit the derivative of the support $B(x)$. We can apply auto-differentiation to compute the derivatives

$$u'(x) = Q'\big(v_L(x)\big) \cdot \sum_{B(x)} \kappa^{(l)'}(x, y, v_{L-1}(y)) \tag{6.13}$$

Similarly, for the Fourier convolution operator, we need to evaluate the Fourier convolution $\mathcal{K}_L v_{L-1}(x)$ on the query points x. It can be done by writing out the output function as Fourier series composing with Q :

$$u(x) = \mathcal{Q} \circ \mathcal{F}^{-1}\Big(R \cdot (\mathcal{F} v_{L-1}))\Big)(x) = Q\Big(\frac{1}{k_{max}} \sum_{k=0}^{k_{max}} \big(R_k (\mathcal{F} v_{L-1})_k\big) \exp \frac{i2\pi k}{D}(x)\Big)$$

where $\mathcal{F}$ is the discrete Fourier transform. The inverse discrete Fourier transform is the sum of k_{max} Fourier series with the coefficients $\big(R_k(\mathcal{F} v_{L-1})_k\big)$ coming from the previous layer.

$$u'(x) = Q'\big(v_L(x)\big) \cdot \frac{1}{k_{max}} \sum_{k=0}^{k_{max}} \big(R_k (\mathcal{F} v_{L-1})_k\big) \exp' \frac{i2\pi k}{D}(x) \tag{6.14}$$

Notice $\exp' \frac{i2\pi k}{D}(x) = \frac{i2\pi k}{D} \exp \frac{i2\pi k}{D}(x)$, just as the numerical Fourier method. If the query points x are a uniform grid, the derivative can be efficiently computed with the Fast Fourier transform.

The autograd method is general and exact, however, it is less efficient. Since the number of parameters $|\theta|$ is usually much greater than the grid size n, the numerical methods are indeed significantly faster. Empirically, the autograd method is usually slower and memory-consuming.

Function-wise differentiation. While it is expensive to apply the auto differentiation per query point, the derivative can be batched and computed in a function-wise manner. We develop an efficient and exact derivatives method based on the architecture of the neural operator that can compute the full gradient field. The idea is similar to the autograd, but we explicitly write out the derivatives on the Fourier space and apply the chain rule. Given the explicit form (6.14), u' can be directly computed on the Fourier space.

$$u' = Q'(v_L) \cdot \mathcal{F}^{-1}\left(\frac{i2\pi}{D} K \cdot (\mathcal{F} v_L)\right) \tag{6.15}$$

Therefore, to exactly compute the derivative of the Fourier neural operator, one just needs to run the numerical Fourier differentiation. Especially the derivative and be efficiently computed with the Fast Fourier transform when the query is uniform. Similarly, if the kernel function $\kappa^{(l)}$ in (6.13) has a structured form, we can also write out its gradient field explicitly.

Next, we show how to compute higher orders derivatives in their exact form, without evoking the autograd method. To this end, we can directly apply the chain rule for the higher-order derivatives without calling autograd. For example, the first order derivatives is $u' = (Q \circ v_L)' = Q'(v_L) \cdot v_L'$ and the 2nd-order derivatives is $u'' = (Qv_L)'' = v_L'^2 \cdot Q''(v_L) + Q'(v_L) \cdot v_L''$. Higher-order derivatives can be similarly computed using the chain rule. Furthermore, derivative-based quantities on v_L, e.g., v_L' can be computed in its exact form in the Fourier domain. Similarly, we can write out the higher-order derivatives of Q using the chain rule. Usually Q is parameterized as a two layer neural networks $Q(x) = (A_2\sigma(A_1 x + b_1) + b_2)$. So $Q'(x) = A_2\sigma'(A_1 x + b_1)A_1$. In this manner, we have got the explicit formula of the derivatives for all neural operators.

Fourier continuation. The Fourier method has its best performance when applied to periodic problems. If the target function is non-periodic or non-smooth, the Fourier differentiation is not accurate. To deal with this issue, we apply the Fourier continuation method that embeds the problem domain into a larger and periodic space. The extension can be simply done by padding zeros in the input. The loss is computed at the original space during training. The Fourier neural operator will automatically generate a smooth extension. The details are given in Appendix 6.3.

Inverse problem

The physics-informed method can be used to solve the inverse problem, where given the output function u, the goal is to recover (a distribution of) the input function a. By imposing the constraint $\mathcal{P}(u, a) = 0$, we can restrict a to a physically valid manifold. We propose two formulations to do the optimization-based inverse problem with PINO: the forward operator model and the inverse operator model.

- **Forward model**: learn the forward operator $\mathcal{G}_\theta : a \mapsto u$ with data. Initialize $\hat{a}$ to approximate $a^\dagger$. Optimize $\hat{a}$ using

$$\mathcal{J}_{forward} \coloneqq \mathcal{L}_{pde}(\hat{a}, u^\dagger) + \mathcal{L}_{data}(\mathcal{G}_\theta(\hat{a})) + R(\hat{a}). \tag{6.16}$$

- **inverse model**: learn the inverse operator $\mathcal{F}_\theta : u \mapsto a$ with data. Use $\mathcal{F}_\theta(u^\dagger)$ to approximate $a^\dagger$. Optimize $\mathcal{F}_\theta$ using

$$\mathcal{J}_{backward} \coloneqq \mathcal{L}_{pde}(\mathcal{F}_\theta(u^\dagger), u^\dagger) + \mathcal{L}_{op}(\mathcal{F}_\theta(u^\dagger), \mathcal{F}_{\theta_0}(u^\dagger)) + R(\mathcal{F}_\theta(u^\dagger) \tag{6.17}$$

where $\mathcal{L}_{pde}$ is the PDE loss; $\mathcal{L}_{op}$ is the operator loss from the learned operator; $R(a)$ is the regularization term. We use the PDE loss $\mathcal{L}_{pde}$ to deal with the small error in $\mathcal{G}_\theta$ and the ill-defining issue of $\mathcal{F}_\theta$. We provide a numerical study in section 6.4.

Fourier continuation

The Fourier neural operator can be applied to arbitrary geometry via Fourier continuations. Given any compact manifold $\mathcal{M}$, we can always embed it into a periodic cube (torus),

$$i : \mathcal{M} \to \mathcal{T}^n$$

where we can do the regular FFT. Conventionally, people would define the embedding i as a continuous extension by fitting polynomials (Bruno, Y. Han, and Pohlman, 2007). However, in Fourier neural operator, it can be simply done by padding zeros in the input. The loss is computed at the original space during training. The Fourier neural operator will automatically generate a smooth extension to do a padded domain in the output, as shown in Figure 6.3.

This technique is first used in the original Fourier neural operator paper (Z. Li, Kovachki, et al., 2021a) to deal with the time dimension in the Navier-Stokes equation. Similarly, (L. Lu, Meng, Cai, et al., 2022) apply FNO with extension

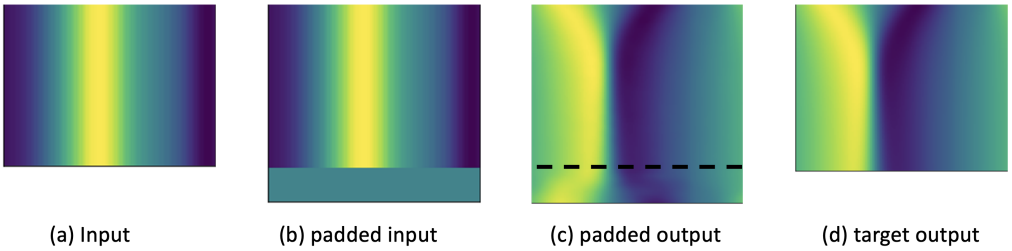


Figure 6.3: Fourier Continuation by padding zeros. The x-axis is the spatial dimension; the y-axis is the temporal dimension. FNO extends the output smoothly on the padded domain.

and interpolation on diverse geometries on the Darcy equation. In the work, we use Fourier continuation widely for non-periodic boundary conditions (Darcy, time dimension). We also added an example of lid-cavity to demonstrate that PINO can work with non-periodic boundary conditions.

Furthermore, this Fourier continuation technique helps to take the derivatives of the Fourier neural operator. Since the output of FNO is always on a periodic domain, the numerical Fourier gradient is usually efficient and accurate, except if there is shock (in this case, we will use the exact gradient method).

6.4 Experiments

In this section, we conduct empirical experiments to examine the efficacy of the proposed PINO. We present the PDE settings, their domains, and function spaces. In 6.4, we show using PDE constraint in operator learning results in neural operators that (1) generalize to significantly high-resolution unseen data. (2) achieve smaller generalization errors with fewer to no data. Then in 6.4, we investigate how PINO uses the operator ansatz to solve harder equations with improved speed and accuracy. We study three concrete cases of PDEs on Burgers' Equation, Darcy's Equation, and Navier-Stokes equation. In 6.4 we study the inverse problems.

Burgers' Equation. The 1-d Burgers' equation is a non-linear PDE with periodic boundary conditions where $u_0 \in L^2_{\text{per}}((0,1);\mathbb{R})$ is the initial condition and $\nu = 0.01$ is the viscosity coefficient. We aim to learn the operator mapping the initial condition to the solution, $\mathcal{G}^\dagger : u_0 \mapsto u|_{[0,1]}$.

$$\begin{aligned} \partial_t u(x,t) + \partial_x (u^2(x,t)/2) &= \nu \partial_{xx} u(x,t), \qquad && x \in (0,1), t \in (0,1] \\ u(x,0) &= u_0(x), && x \in (0,1) \end{aligned} \tag{6.18}$$

Darcy Flow. The 2-d steady-state Darcy Flow equation on the unit box which is the second order linear elliptic PDE with a Dirichlet boundary where $a \in L^\infty((0,1)^2;\mathbb{R}_+)$ is a piecewise constant diffusion coefficient and $f = 1$ is a fixed forcing function. We are interested in learning the operator mapping the diffusion coefficient to the solution, $\mathcal{G}^\dagger : a \mapsto u$. Note that although the PDE is linear, the operator $\mathcal{G}^\dagger$ is not.

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x) \qquad && x \in (0,1)^2 \\ u(x) &= 0 && x \in \partial(0,1)^2 \end{aligned} \tag{6.19}$$

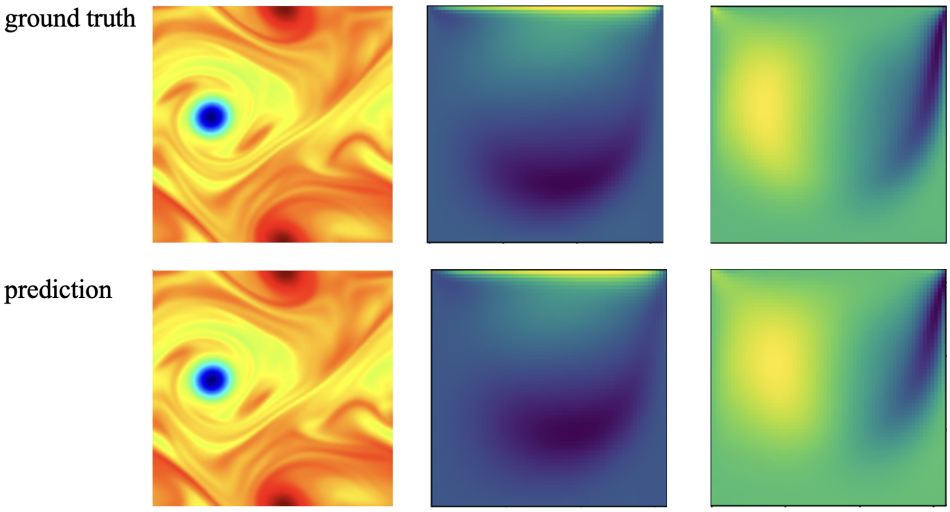
Since a is in L_{inf}, we considered both the strong form $\mathcal{L}_{pde}(u) = \nabla \cdot (a\nabla u) - f$ and the weak form minimization loss $\mathcal{L}_{pde}(u) = -\frac{1}{2}(a\nabla u, \nabla u) - (u, f), u \in H_1$. Experiments show the strong form has a better performance.

Navier-Stokes Equation. We consider the 2-d Navier-Stokes equation for a viscous, incompressible fluid in vorticity form on the unit torus, where $u \in C([0,T]; H^r_{\text{per}}((0,l)^2;\mathbb{R}^2))$ for any $r > 0$ is the velocity field, $w = \nabla \times u$ is the vorticity, $w_0 \in L^2_{\text{per}}((0,l)^2;\mathbb{R})$ is the initial vorticity, $\nu \in \mathbb{R}_+$ is the viscosity coefficient, and $f \in L^2_{\text{per}}((0,l)^2;\mathbb{R})$ is the forcing function. We want to learn the operator mapping the vorticity from the initial condition to the full solution $\mathcal{G}^\dagger : w_0 \mapsto w|_{[0,T]}$.

$$\begin{aligned} \partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) &= \nu \Delta w(x,t) + f(x), && x \in (0,l)^2, t \in (0,T] \\ \nabla \cdot u(x,t) &= 0, && x \in (0,l)^2, t \in [0,T] \\ w(x,0) &= w_0(x), && x \in (0,l)^2 \end{aligned} \tag{6.20}$$

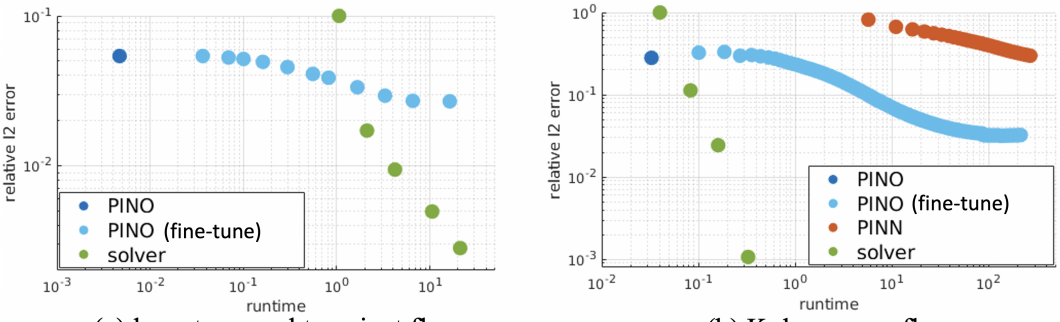
Specially, we consider two problem settings:

- **Long temporal transient flow**: we study the build-up of the flow from the initial condition u_0 near-zero velocity to u_T that reaches the ergodic state. We choose $t \in [0, 50]$, $l = 1$, $Re = 20$ as in (Z. Li, Kovachki, et al., 2021a). The main challenge is to predict the long time interval.
- **Chaotic Kolmogorov flow**: In this case u lies in the attractor where arbitrary starting time t_0. We choose $t \in [t_0, t_0 + 0.5]$ or $[t_0, t_0 + 1]$, $l = 2\pi$, $Re = 500$ similar to (Z. Li, Kovachki, et al., 2023). The main challenge is to capture the small details that evolve chaotically.



(a) KF: vorticity (b) Cavity: velocity in x, y

Figure 6.4: PINO on Kolmogorov flow (left) and Lid-cavity flow (right)



(a) long-temporal transient flow (b) Kolmogorov flow

(a) The long-temporal transient flow with $Re \sim 20, T = 50$. PINO outputs the full trajectory in one step, which leads to a 400x speedup compared to the GPU solver. PINN cannot converge to a reasonable error rate due to the long time window. (b) The chaotic Kolmogorov flow with $Re =\sim 500, T = 0.5$. PINO converges faster compared to PINN, but their convergence rates with gradient descent are less effective compared to using higher resolutions in the GPU solver.

Figure 6.5: The accuracy-complexity trade-off on PINO, PINN, and the GPU-based pseudo-spectral solver.

- **Lid cavity flow**: In this case, we assume the no-slip boundary condition where $u(x,t) = (0,0)$ at left, bottom, and right walls and $u(x,t) = (1,0)$ on top, similar to (Bruneau and Saad, 2006). We choose $t \in [5,10]$, $l = 1$, $Re = 500$. The main challenge is to address the boundary using the velocity-pressure formulation.

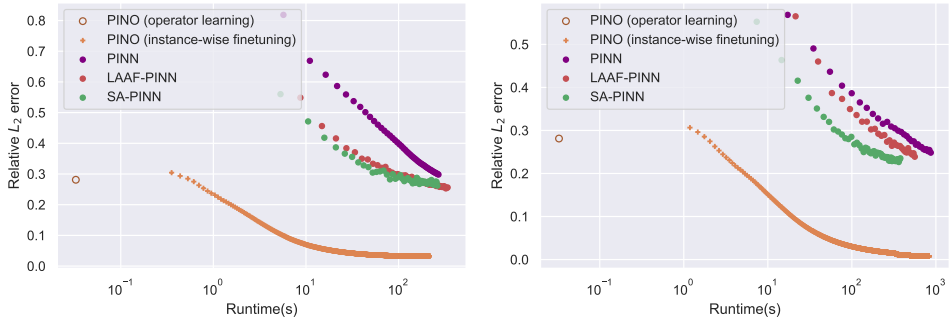


Figure 6.6: Plot of test relative L_2 error versus runtime step for the Kolmogorov flow with Re500, T=0.5s. Left: resolution $64\times64\times65$; right: resolution $128\times128\times129$. Averaged over 20 instances. LAAF-PINN: PINN with locally adaptive activation functions. SA-PINN: self-adaptive PINN.

Operator learning with physics constraints

We show that we can utilize the equation constraints to improve neural operator training. For this purpose, we train neural operators on fixed-resolution data in the presence of physics loss, $\mathcal{J}_{pde}$, and test the performance of the trained operators on high-resolution data. In particular, we test the performance of the trained model on data with the same resolution of the training data, 1x, 2x, and 4x, of the training data resolution Table 6.1. We observe that incorporating the $\mathcal{J}_{pde}$ in the training results in operators that, with high accuracy, generalize across data resolution. In this experiment, the training data for Burgers equation setting is in 32×25 (spatio-temporal), and the $\mathcal{J}_{pde}$ is imposed in 128×100 resolution. We use 800 low-resolution data and the same 800 PDE instances. The mean relative L_2 error and its standard deviation are reported over 200 test instances at resolution 32×25, 64×50, and 128×100.

Accordingly, the training data for the Darcy equation setting is at the spatial resolution of 11×11 and the $\mathcal{J}_{pde}$ is imposed in 61×61 resolution. We use 1000 low-resolution data and the same 1000 PDE instances. The mean and standard error are reported over 500 test instances at resolution 11×11, 61×61, and 211×211. Darcy flow is unresolved at the 11×11 resolution, training on such a coarse grid causes higher errors. However, adding higher resolution PDE loss helps the operator to resolve.

The training data for Kolmogorov flow is in $64\times64\times33$ and the $\mathcal{J}_{pde}$ is imposed in $256\times256\times65$ resolution for the time interval $[0, 0.125]$. We use 800 low-resolution data and 2200 PDE instances. The mean and std of the relative L_2 error are reported over 200 test instances, at resolution $64\times64\times33$, $128\times128\times33$, and $256\times256\times65$.

PDE	Training setting	Error at low data resolution	Error at 2× data resolution	Error at 4× data resolution
Burgers	Data	0.32±0.01%	3.32±0.02%	3.76±0.02%
	Data & PDE loss	0.17±0.01%	0.28±0.01%	0.38±0.01%
Darcy	Data	5.41±0.12%	9.01±0.07%	9.46±0.07%
	Data & PDE loss	5.23±0.12%	1.56±0.05%	1.58±0.06%
Kolm. flow	Data	8.28%±0.15%	8.27%±0.15%	8.30%±0.15%
	Data & PDE loss	6.04%±0.12%	6.02%±0.12%	6.01%±0.12%

Table 6.1: Operator-learning using fixed resolution data and PDE loss allows us to train operators with high accuracy on higher resolution unseen data.

Burgers equation and Darcy equation. PINO can learn the solution operator without any data on simpler problems such as Burgers and Darcy. Compared to other PDE-constrained operators, PINO is more expressive and thereby achieves better accuracy. On Burgers (6.18), PI-DeepONet achieves **1.38%** (S. Wang, H. Wang, and Perdikaris, 2021); PINO achieves **0.38%**. Similarly, on Darcy flow (7.2), PINO outperforms FNO by utilizing physics constraints, as shown in Table 6.2. For these simpler equations, instance-wise fine-tuning may not be needed.

Method	Solution error
DeepONet with data	6.97 ± 0.09%
PINO with data	1.22 ± 0.03%
PINO w/o data	1.50 ± 0.03%

Table 6.2: Operator learning on Darcy Flow equation. Incorporating physics constraints in operator learning improves the performance of neural operators.

# data samples	# PDE instances	Solution error
0	2200	6.22%±0.11%
800	2200	6.01%±0.12%
2200	2200	5.04%±0.11%

Table 6.3: Physics-informed neural operator learning on Kolmogorov flow $Re = 500$. PINO is effective and flexible in combining physics constraints and any amount of available data. The mean and standard error of the relative L_2 test error is reported over 200 instances and evaluated on resolution $256 \times 256 \times 65$.

Chaotic Kolmogorov flow. We conduct an empirical study on how PINO can improve the generalization of neural operators by enforcing more physics. In the first experiment, we consider the Kolmogorov flow with $T = 0.125$. We train PINO

with 2200 initial conditions and different amounts of low-resolution data. As shown in Table 6.3, PINO achieves 6.22% error even without any data. We also observe that adding more low-resolution data to training makes the optimization easier and consistently improves the accuracy of the learned operator, showing that PINO is effective and flexible in combining physics constraints and any amount of available data.

The second experiment considers the Kolmogorov flow with $T = 0.5$. The training set consists of 4000 data points of the initial condition and corresponding solution. For operator learning, we sample high-resolution initial conditions from a Gaussian random field. Table 6.7 compare the generalization error of neural operators trained by different schemes and different amounts of simulated data. The result shows that training neural operator with additional PDE instances consistently improves the generalization error on all three resolutions we are evaluating. Note that the relative L_2 error in this setting is much higher than the first one because the time horizon is 4 times longer. Next, we show how to solve for specific instances by finetuning the learned operator.

Solve equation using operator ansatz

We solve specific equation instances by fine-tuning the learned operator ansatz.

Long temporal transient flow. It is extremely challenging to propagate the information from the initial condition to future time steps over such a long interval $T = [0, 50]$ just using the soft physics constraint. Neither the PINN nor PINO (from scratch) can handle this case (error $> 50\%$), no matter solving the full interval at once or solving per smaller steps. However, when the data is available for PINO, we can use the learned neural operator ansatz and the anchor loss $\mathcal{L}_{op}$. The anchor loss is a hard constraint that makes the optimization much easier. Providing $N = 4800$ training data, the PINO without instance-wise fine-tuning achieves **2.87%** error, lower than FNO **3.04%** and it retains a 400x speedup compared to the GPU-based pseudo-spectral solver (He and W. Sun, 2007), matching FNO. Further doing test time optimization with the anchor loss and PDE loss, PINO reduces the error to **1.84%**.

Chaotic Kolmogorov flow. Based on the solution operators learned in Section 6.4, the second operator-learning setting, we continue to do instance-wise fine-tuning. We compare our method against other physics-informed learning methods including

PINN (Raissi, Perdikaris, and George E Karniadakis, 2019), LAAF-PINN (Jagtap, Kawaguchi, and Em Karniadakis, 2020), and SA-PINN (McClenny and Braga-Neto, 2023), as shown in Figure 6.6 and Table 6.4. Overall, PINO outperforms PINN and its improved variants by **20x** smaller error and **25x** speedup. Using a learned operator model makes PINO converge faster.

Method	# data samples	# PDE instances	Solution error (w)	Time cost
PINNs	-	-	18.7%	4577s
PINO	0	0	**0.9%**	608s
PINO	0.4k	0	**0.9%**	536s
PINO	0.4k	160k	**0.9%**	**473**s

Table 6.4: Instance-wise fine-tuning on Kolmogorov flow $Re = 500, T = 0.5$. Using the learned operator as the initial condition helps fine-tuning converge faster.

Zero-shot super-resolution. The neural operator models are discretization-convergent, meaning they can take the training dataset of variant resolutions and be evaluated at higher resolution. As shown in Figure 6.1, we train the FNO, PINO, and UNet model with $64 \times 64 \times 32$ Kolmogorov Flows dataset and evaluate them at $256 \times 256 \times 65$ resolution. Any frequencies higher than 64 are unseen during the training time. Conventional models such as UNet are not capable of direct super-resolution. For compassion, we equip it with tri-linear interpolation. For PINO, we also do test-time optimization. As shown in Figure 6.1, the spectrums of the predictions are averaged over 50 instances. PINO with the test-time optimization achieves a very high accuracy rate, and its spectrum overlaps with the ground truth spectrum. However, Conventional models such as UNet+Interpolation have noising prediction with oscillating high frequencies. On the other hand, with the help of test-time optimization, PINO can extrapolate to unseen frequencies with high accuracy.

Transfer Reynolds numbers. The extrapolation of different parameters and conditions is one of the biggest challenges for ML-based methods. It poses a domain shift problem. In this experiment, we train the source operator model on one Reynolds number and then fine-tune the model to another Reynolds number, on the Kolmogorov flow with $T = 1$. As shown in Table 6.5 by doing instance-wise fine-tuning, PINO can be easily transferred to different Reynolds numbers ranging from 100 to 500. This transferability shows PINO learned the dynamics shared across different Reynolds numbers. Such property envisions broad applications including transferring the learned operator to different boundary conditions or geometries.

Transfer learning across Reynolds numbers

We study the instance-wise fine-tuning with different Reynolds numbers on the $T = 1$ Kolmogorov flow. For the higher Reynolds number problem $Re = 500, 400$, fine-tuning the source operator shows better convergence accuracy than learning from scratch. In all cases, the fine-tuning of the source operator shows better convergence speed as demonstrated in Figure 6.7. The results are shown in Table 6.5 where the error is averaged over 40 instances. Each row is a testing case, and each column is a source operator.

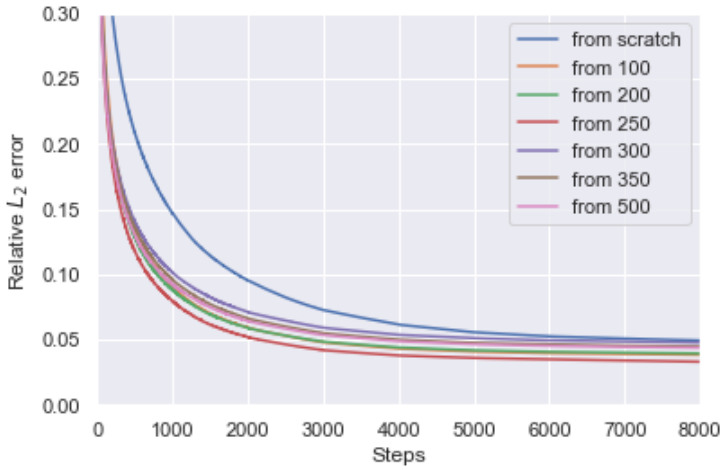
Testing Re	From scratch	100	200	250	300	350	400	500
500	4.93	3.83	3.93	3.15	4.77	4.46	4.34	4.36
400	2.96	2.43	2.45	2.44	3.00	2.71	2.73	2.40
350	1.92	2.10	2.11	2.13	2.33	2.22	2.22	2.12
300	1.68	1.61	1.64	1.51	1.77	1.73	1.70	1.60
250	1.51	1.50	1.53	1.51	1.60	1.56	1.60	1.51
200	0.921	0.913	0.921	0.915	0.985	0.945	0.923	0.892
100	0.234	0.235	0.236	0.235	0.239	0.239	0.237	0.237

Each row is a test set of PDEs with corresponding Reynolds number. Each column represents the operator ansatz we use as the starting point of instance-wise fine-tuning. For example, column header "100" means the operator ansatz is trained over a set of PDEs with Reynolds number 100. The relative L_2 errors is averaged over 40 instances of the corresponding test set.

Table 6.5: Reynolds number transfer learning.

Lid cavity flow. We demonstrate an additional example using PINO to solve for lid-cavity flow on $T = [5, 10]$ with $Re = 500$. In this case, we do not have the operator-learning phase and directly solve the equation (instance-wise fine-tuning). We use PINO with the velocity-pressure formulation and resolution $65 \times 65 \times 50$ plus the Fourier numerical gradient. It takes 2 minutes to achieve a relative error of 14.52%. Figure 6.4 shows the ground truth and prediction of the velocity field at $t = 10$ where the PINO accurately predicts the ground truth. The experiment shows that PINO can address non-periodic boundary conditions and multiple output fields.

Convergence of accuracy with respect to resolution. We study the convergence rate of PINO in the instance-wise optimization setting, where we minimize the PDE loss under different resolutions without any data. For PINO, using a higher resolution is more effective compared to running gradient descent for longer iterations. We test PINO on the Kolmogorov flow with Re = 500 and T = 0.125. We use the Fourier method in the spatial dimension and the finite difference method in the



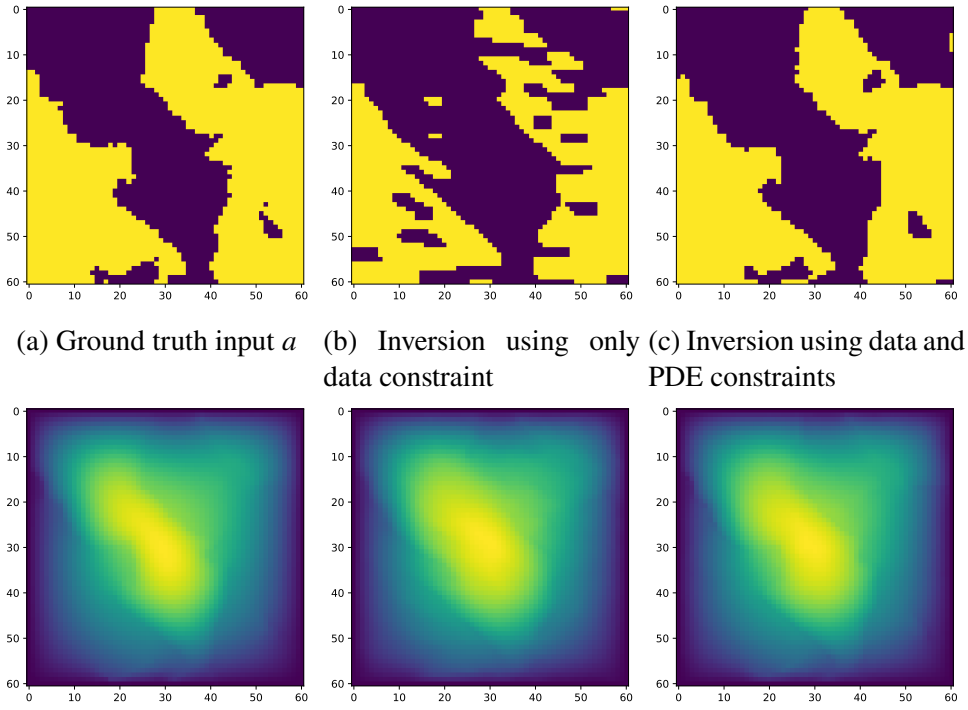
The test error is averaged over 40 instances. We observe that all the operator ansatzs trained over PDE instances with different Reynolds numbers can boost the instance-wise fine-tuning accuracy and convergence speed compared to training from scratch.

Figure 6.7: Plot of relative L_2 error versus update step for the Kolmogorov flow with Reynolds number 500, $T = 1$.

temporal dimension. As shown in Table 6.6, PINO shares the same convergence rate of its differentiation methods with no obvious limitation from optimization. It has an exponential convergence rate in space and a first-order convergence rate in time $err = O(exp(dx)) + O(dt)$. It implies the PDE constraint can achieve high accuracy given a reasonable computational cost, and the virtual instances are almost as good as the data instances generated by the solver. Since the PDE loss can be computed on an unlimited amount of virtual instances in the operator learning setting, it is possible to reduce the generalization error going to zero by sampling virtual instances.

dx \ dt	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}
2^{-4}	0.4081	0.3150	0.3149	0.3179	0.3196
2^{-5}	0.1819	0.1817	0.1780	0.1773	0.1757
2^{-6}	0.0730	0.0436	0.0398	0.0386	0.0382
2^{-7}	0.0582	0.0234	0.0122	0.0066	0.0034

Table 6.6: relative L2 error of PINO (Finite-difference in time) on Kolmogorov flow with $Re = 500$ and $T = 0.125$. PINO inherits the convergence rate of its differentiation method with no limitation of optimization.



(a) Ground truth input a (b) Inversion using only data constraint (c) Inversion using data and PDE constraints

(d) Observed output function (e) Output function of inversion using only data constraint (f) Output function of inversion using data and PDE constraints

In the above figures, (6.8a) represents the ground truth input function $a^\dagger$, and (6.8d) demonstrates the corresponding solution $u^\dagger$, i.e., the output function. Given the output $u^\dagger$, we aim to recover what a could have generated the output function $u^\dagger$. Using only data constraint, (6.8b) shows that our method can find an a that results in an output function very close to the ground truth $u^\dagger$ (6.8e). However, the recovered a is far from satisfying the PDE equation. Using both data and PDE constraints, (6.8c) shows that our physics-informed method can find an a that not only results in an output function very close to the ground truth $u^\dagger$(6.8f), but also the recovered a satisfies the PDE constraint and is close to the underlying $a^\dagger$.

Figure 6.8: Inverse Problem with Physics Informed Neural Operator

Inverse problem

One of the major advantages of the physics-informed method is to solve the inverse problem. In the experiment, we investigate PINO on the inverse problem of the Darcy equation to recover the coefficient function $a^\dagger$ from the given solution function $u^\dagger$. We assume a dataset $\{a_j, u_j\}$ is available to learn the operator. The coefficient function a is a piecewise constant (representing two types of media), so the inverse problem can be viewed as a classification problem. We define $R(a)$ as the total variance.

The PDE loss strongly improves the prediction of the inverse problem. The plain

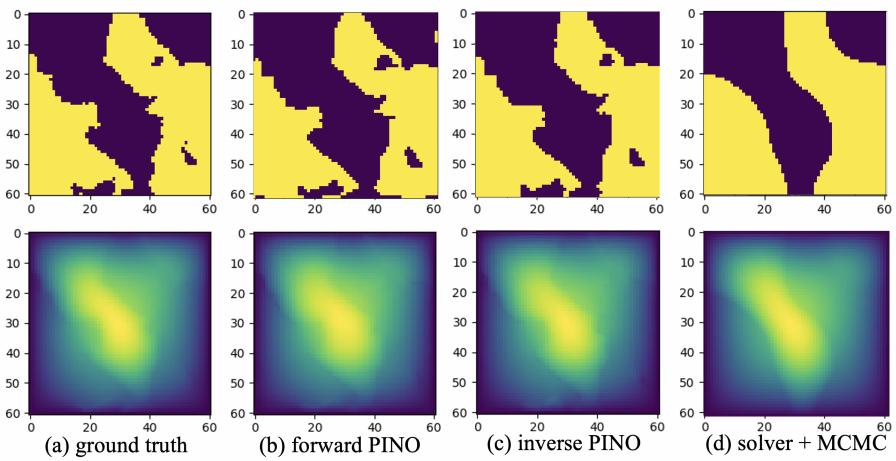


Figure 6.9: Darcy inverse problem: comparing PINO forward, inverse models with numerical solver with MCMC.

neural operator model, while accurate in the forward problem, is vulnerable under perturbation and shift of the input a, which is a common behavior of deep-learning models. This domain-shift problem is critical for optimization-based inverse problems. During the optimization, a is likely to go out of the training distribution, which makes the neural operator model inaccurate. As shown in Figure 6.8 (b), the prediction of a is less accurate, while the model believes its output 6.8 (e) is the same as the target. This issue is mitigated by adding the PDE constraints, which restrict the prediction a to the physically-valid manifold where $\mathcal{P}(a, u) = 0$. As shown in Figure 6.8 (c), the initial condition recovered with PDE loss is very close to the ground truth.

Comparing the PINO forward model with the inverse model, the inverse model $\mathcal{F}_\theta : u \mapsto a$ (6.17) has the best performance. As Shown in Figure 6.9, the inverse model has **2.29%** relative l2 error on the output u and **97.10%** classification accuracy on the input a; the forward model has 6.43% error on the output and 95.38% accuracy on the input. Both models converge with 200 iterations. The major advantage of the PINO inverse model compared to the PINO forward model is that it uses a neural operator $\mathcal{F}_\theta(u^\dagger)$ as the ansatz for the coefficient function, which is used as regularization $\mathcal{L}_{op}$. Similar to the forward problem, the operator ansatz has an easier optimization landscape while being expressive.

As a reference, we compare the PINO inverse frameworks with PINN and the conventional solvers using the accelerated MCMC method with 500,000 steps (Cotter et al., 2013). The posterior mean of the MCMC has a 4.52% error and 90.30% respectively (Notice the Bayesian method outputs the posterior distribution, which is

beyond obtaining a maximum a posteriori estimation). Meanwhile, PINO methods are 3000x faster compared to MCMC PINN does not converge in this case.

Besides the speedup with respect to the online cost, the offline training of PINO only takes around 1 hour on a single GPU on the Darcy problem. Once trained, the model can be used without any further training cost. As a comparison, it takes considerably longer to deploy a finite element solver and an MCMC solver compared to a machine learning model. Generally speaking, numerical solvers usually have a more complicated codebase and it is non-trivial to specify boundary conditions, time schemes, and meshes. In the end, it can be easier to prepare a machine learning model than a standard numerical solver. Flexibility and accessibility are some of the major advantages of these machine learning methods.

# Data	# PDE	Resolution	Solution error	Equation error
400	0	$128 \times 128 \times 65$	33.32%	1.8779
		$64 \times 64 \times 65$	33.31%	1.8830
		$32 \times 32 \times 33$	30.61%	1.8421
400	40k	$128 \times 128 \times 65$	31.74%	1.8179
		$64 \times 64 \times 65$	31.72%	1.8227
		$32 \times 32 \times 33$	29.60%	1.8296
400	160k	$128 \times 128 \times 65$	31.32%	1.7840
		$64 \times 64 \times 65$	31.29%	1.7864
		$32 \times 32 \times 33$	29.28%	1.8524
4k	0	$128 \times 128 \times 65$	25.15%	1.8223
		$64 \times 64 \times 65$	25.16%	1.8257
		$32 \times 32 \times 33$	21.41%	1.8468
4k	100k	$128 \times 128 \times 65$	24.15%	1.6112
		$64 \times 64 \times 65$	24.11%	1.6159
		$32 \times 32 \times 33$	20.85%	1.8251
4k	400k	$128 \times 128 \times 65$	24.22%	1.4596
		$64 \times 64 \times 65$	23.95%	1.4656
		$32 \times 32 \times 33$	20.10%	1.9146
0	100k	$128 \times 128 \times 65$	74.36%	0.3741
		$64 \times 64 \times 65$	74.38%	0.3899
		$32 \times 32 \times 33$	74.14%	0.5226

Table 6.7: Each neural operator is trained with 400 or 4000 data points additionally sampled free initial conditions. The Reynolds number is 500. The reported generalization error is averaged over 300 instances. Training on additional initial conditions boosts the generalization ability of the operator.

6.5 Discussion and Conclusion

In this work, we develop the physics-informed neural operator (PINO) that bridges the gap between physics-informed optimization and data-driven neural operator learning. We introduce operator-learning and instance-wise fine-tuning schemes for PINO to utilize both the data and physics constraints. In the operator learning phase, PINO learns an operator ansatz over multiple instances of a parametric PDE family. The instance-wise fine-tuning scheme allows us to take advantage of the learned neural operator ansatz and solve for the solution function on the querying instance faster and more accurately.

While PINO shows many promising applications, it also shares some limitations as in the previous work. For example, since PINO is currently implemented with the FNO backbone with the Fast-Fourier transform, it is hard to extend to higher dimensional problems. Besides, as shown in Figure 6.5, finetuning PINO using gradient descent methods does not converge as fast as using a finer grid as in Table 6.6. Further optimization techniques are to be developed.

There are many exciting future directions. Most of the techniques and analyses of PINN can be transferred to PINO. It is also interesting to ask how to overcome the hard trade-off of accuracy and complexity, and how the PINO model transfers across different geometries. Furthermore, it is promising to develop a software library of pre-trained models. PINO's excellent extrapolation property allows it to be applied on a broad set of conditions, as shown in Transfer Reynold's number experiments.

References

Bartolucci, Francesca et al. (2023). "Representation Equivalent Neural Operators". In: *Conference on Neural Information Processing Systems*.

Bhattacharya, Kaushik et al. (2021). "Model reduction and neural networks for parametric PDE(s)". In: *The SMAI Journal of computational mathematics, Volume 7, pp. 121-157.*

Bonev, Boris et al. (2023). "Spherical Fourier Neural Operators: Learning Stable Dynamics on the Sphere". In: *arXiv preprint arXiv:2306.03838*.

Bruneau, Charles-Henri and Mazen Saad (2006). "The 2D lid-driven cavity problem revisited". In: *Computers & fluids* 35.3, pp. 326–348.

Bruno, Oscar P, Youngae Han, and Matthew M Pohlman (2007). "Accurate, high-order representation of complex three-dimensional surfaces via Fourier continuation analysis". In: *Journal of computational Physics* 227.2, pp. 1094–1125.

Brunton, Steven L, Bernd R Noack, and Petros Koumoutsakos (2020). "Machine learning for fluid mechanics". In: *Annual Review of Fluid Mechanics* 52, pp. 477–508.

Cai, Shengze et al. (2021). "Physics-informed neural networks (PINNs) for fluid mechanics: A review". In: *Acta Mechanica Sinica*.

Cotter, Simon L et al. (2013). "MCMC methods for functions: modifying old algorithms to make them faster". In: *Statistical Science* 28.3, pp. 424–446.

Duvall, James, Karthik Duraisamy, and Shaowu Pan (2021). *Non-linear Independent Dual System (NIDS) for Discretization-independent Surrogate Modeling over Complex Geometries*. arXiv: 2109.07018 [physics.comp-ph].

Dwivedi, Vikas and Balaji Srinivasan (2020). "Physics Informed Extreme Learning Machine (PIELM)–A rapid method for the numerical solution of partial differential equations". In: *Neurocomputing* 391, pp. 96–118.

Fuks, Olga and Hamdi A Tchelepi (2020). "Limitations of physics informed machine learning for nonlinear two-phase transport in porous media". In: *Journal of Machine Learning for Modeling and Computing* 1.1.

Gao, Han, Luning Sun, and Jian-Xun Wang (2021). "PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain". In: *Journal of Computational Physics* 428, p. 110079.

Greenfeld, Daniel et al. (2019). "Learning to optimize multigrid PDE solvers". In: *International Conference on Machine Learning*. PMLR, pp. 2415–2423.

Han, Jiequn, Arnulf Jentzen, and E Weinan (2018). "Solving high-dimensional partial differential equations using deep learning". In: *Proceedings of the National Academy of Sciences* 115.34, pp. 8505–8510.

He, Yinnian and Weiwei Sun (2007). "Stability and convergence of the Crank–Nicolson/Adams–Bashforth scheme for the time-dependent Navier–Stokes equations". In: *SIAM Journal on Numerical Analysis* 45.2, pp. 837–869.

Hennigh, Oliver et al. (2021). "NVIDIA SimNet™: An AI-accelerated multi-physics simulation framework". In: *International Conference on Computational Science*. Springer, pp. 447–461.

Hersbach, Hans et al. (2020). "The ERA5 global reanalysis". In: *Quarterly Journal of the Royal Meteorological Society* 146.730, pp. 1999–2049.

Hoop, Maarten V de et al. (2023). "The cost-accuracy trade-off in operator learning with neural networks". In: *Journal of Machine Learning*.

Huang, Xiang et al. (2022). "Meta-Auto-Decoder for Solving Parametric Partial Differential Equations". In: *Conference on Neural Information Processing Systems*.

Jagtap, Ameya D, Kenji Kawaguchi, and George Em Karniadakis (2020). "Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks". In: *Proceedings of the Royal Society A* 476.2239, p. 20200334.

Jin, Xiaowei et al. (2021). "NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations". In: *Journal of Computational Physics* 426, p. 109951.

Kashinath, K et al. (2021). "Physics-informed machine learning: case studies for weather and climate modelling". In: *Philosophical Transactions of the Royal Society A* 379.2194, p. 20200093.

Kochkov, Dmitrii et al. (2021). "Machine learning accelerated computational fluid dynamics". In: *Proceedings of the National Academy of Sciences (PNAS)*.

Kontolati, Katiana et al. (2023). "On the influence of over-parameterization in manifold based surrogates and deep neural operators". In: *Journal of Computational Physics* 479, p. 112008.

Kovachki, Nikola et al. (n.d.). "Neural operator: Learning maps between function spaces". In: *Journal of Machine Learning Research. volume 24, number 89, pages 1-97. (2023)* ().

Lanthaler, Samuel et al. (2023). "Nonlinear Reconstruction for Operator Learning of PDEs with Discontinuities". In: *International Conference on Learning Representations*.

Li, Yingzhou, Jianfeng Lu, and Anqi Mao (2020). "Variational training of neural network approximations of solution maps for physical models". In: *Journal of Computational Physics* 409, p. 109338.

Li, Zongyi, Daniel Zhengyu Huang, et al. (n.d.). "Fourier neural operator with learned deformations for pdes on general geometries". In: *Journal of Machine Learning Research (388):1-26.(2023)* ().

Li, Zongyi, Nikola Kovachki, et al. (2021a). "Fourier Neural Operator for Parametric Partial Differential Equations". In: *International Conference on Learning Representations*.

– (2021b). "Multipole Graph Neural Operator for Parametric Partial Differential Equations". In: *Conference on Neural Information Processing Systems*.

– (2023). "Markov Neural Operators for Learning Chaotic Systems". In: *Conference on Neural Information Processing Systems*.

– (n.d.). "Neural operator: Graph kernel network for partial differential equations". In: *Journal of Machine Learning Research. volume 24, number 89, pages 1-97. (2023)* ().

Liu, Burigede et al. (2022). "A learning-based multiscale method and its application to inelastic impact problems". In: *Journal of the Mechanics and Physics of Solids* 158, p. 104668.

Lu, Denghui et al. (2021). “86 PFLOPS Deep Potential Molecular Dynamics simulation of 100 million atoms with ab initio accuracy”. In: *Computer Physics Communications* 259, p. 107624.

Lu, Lu, Pengzhan Jin, and George Em Karniadakis (2021). “DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators”. In: *Nat Mach Intell 3, 218–229*.

Lu, Lu, Xuhui Meng, Shengze Cai, et al. (2022). “A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data”. In: *Computer Methods in Applied Mechanics and Engineering*.

Lu, Lu, Xuhui Meng, Zhiping Mao, et al. (2021). “DeepXDE: A deep learning library for solving differential equations”. In: *SIAM Review* 63.1, pp. 208–228. DOI: 10.1137/19M1274067.

McClenny, Levi and Ulisses Braga-Neto (2023). “Self-adaptive physics-informed neural networks using a soft attention mechanism”. In: *Journal of Computational Physics, 2023*.

Nelsen, NH and AM Stuart (2021). “The Random Feature Model for Input-Output Maps between Banach Spaces”. In: *SIAM Journal on Scientific Computing*.

Patel, Ravi G et al. (2021). “A physics-informed operator regression framework for extracting data-driven continuum models”. In: *Computer Methods in Applied Mechanics and Engineering* 373, p. 113500.

Pathak, Jaideep, Mustafa Mustafa, et al. (2021). “ML-PDE: A Framework for a Machine Learning Enhanced PDE Solver”. In: *Bulletin of the American Physical Society*.

Pathak, Jaideep, Shashank Subramanian, et al. (2023). “Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators”. In.

Raissi, Maziar, Paris Perdikaris, and George E Karniadakis (2019). “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378, pp. 686–707.

Raissi, Maziar, Alireza Yazdani, and George Em Karniadakis (2020). “Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations”. In: *Science* 367.6481, pp. 1026–1030.

Sirignano, Justin and Konstantinos Spiliopoulos (2018). “DGM: A deep learning algorithm for solving partial differential equations”. In: *Journal of computational physics* 375, pp. 1339–1364.

Smith, Jonathan D et al. (2021). “HypoSVI: Hypocenter inversion with Stein variational inference and Physics Informed Neural Networks”. In: *AGU Fall Meeting 2021*.

Sun, Luning et al. (2020). “Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data”. In: *Computer Methods in Applied Mechanics and Engineering* 361, p. 112732.

Wang, Rui et al. (2020). “Towards physics-informed deep learning for turbulent flow prediction”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1457–1466.

Wang, Sifan, Yujun Teng, and Paris Perdikaris (2021). “Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks”. In: *SIAM Journal on Scientific Computing* 43.5, A3055–A3081.

Wang, Sifan, Hanwen Wang, and Paris Perdikaris (2021). “Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets”. In: *Science advances*.

Wang, Sifan, Xinling Yu, and Paris Perdikaris (2022). “When and why pinns fail to train: A neural tangent kernel perspective”. In: *Journal of Computational Physics, Volume 449*.

Weinan, E and Bing Yu (2018). “The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems”. In: *Communications in Mathematics and Statistics* 6.1, pp. 1–12.

Wen, Gege et al. (2023a). “Real-time high-resolution CO 2 geological storage prediction using nested Fourier neural operators”. In: *Energy & Environmental Science* 16.4, pp. 1732–1741.

– (2023b). “Real-time high-resolution CO2 geological storage prediction using nested Fourier neural operators”. In: *Energy Environ. Sci., 16, 1732-1741*.

Yang, Haoyu et al. (2022). “Generic Lithography Modeling with Dual-band Optics-Inspired Neural Networks”. In: *DAC ’22: Proceedings of the 59th ACM/IEEE Design Automation Conference*.

Zhang, Lulu et al. (2022). “MOD-Net: A Machine Learning Approach via Model-Operator-Data Network for Solving PDEs”. In: *Commun. Comput. Phys.*

Zhu, Yinhao et al. (2019). “Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data”. In: *Journal of Computational Physics* 394, pp. 56–81.

Chapter 7

LEARNING: SCALE CONSISTENCY LEARNING FOR NEURAL OPERATOR

Machine learning (ML) models have emerged as a promising approach for solving partial differential equations (PDEs) in science and engineering. Previous ML models typically cannot generalize outside the training data; for example, a trained ML model for the Navier-Stokes equations only works for a fixed Reynolds number (Re) on a pre-defined domain. To overcome these limitations, we propose a data augmentation scheme based on scale-consistency properties of PDEs and design a scale-informed neural operator that can model a wide range of scales. Our formulation leverages the facts: (i) PDEs can be rescaled, or more concretely, a given domain can be re-scaled to unit size, and the parameters and the boundary conditions of the PDE can be appropriately adjusted to represent the original solution, and (ii) the solution operators on a given domain are consistent on the sub-domains. We leverage these facts to create a scale-consistency loss that encourages matching the solutions evaluated on a given domain and the solution obtained on its sub-domain from the rescaled PDE. Since neural operators can fit to multiple scales and resolutions, they are the natural choice for incorporating scale-consistency loss during training of neural PDE solvers. We experiment with scale-consistency loss and the scale-informed neural operator model on the Burgers' equation, Darcy Flow, Helmholtz equation, and Navier-Stokes equations. With scale-consistency, the model trained on Re of 1000 can generalize to Re ranging from 250 to 10000, and reduces the error by 38% on average of all datasets compared to baselines.

7.1 Introduction

ML for PDEs. Data-driven methods have become increasingly popular in learning Partial Differential Equations (PDEs) for scientific computing (Azizzadenesheli et al., 2024), showing various applications ranging from weather forecasting (Pathak et al., 2022) to nuclear fusion (Gopakumar et al., 2023). While conventional models are typically parameterized for a fixed resolution at a predefined scale, neural operators have recently been proposed to generalize across discretization by parameterizing the model on function spaces (Li, N. Kovachki, et al., 2020b; Nikola B Kovachki et al., 2023; Lu et al., 2021; Kissas et al., 2022; Raonic et al., 2023). Among these,

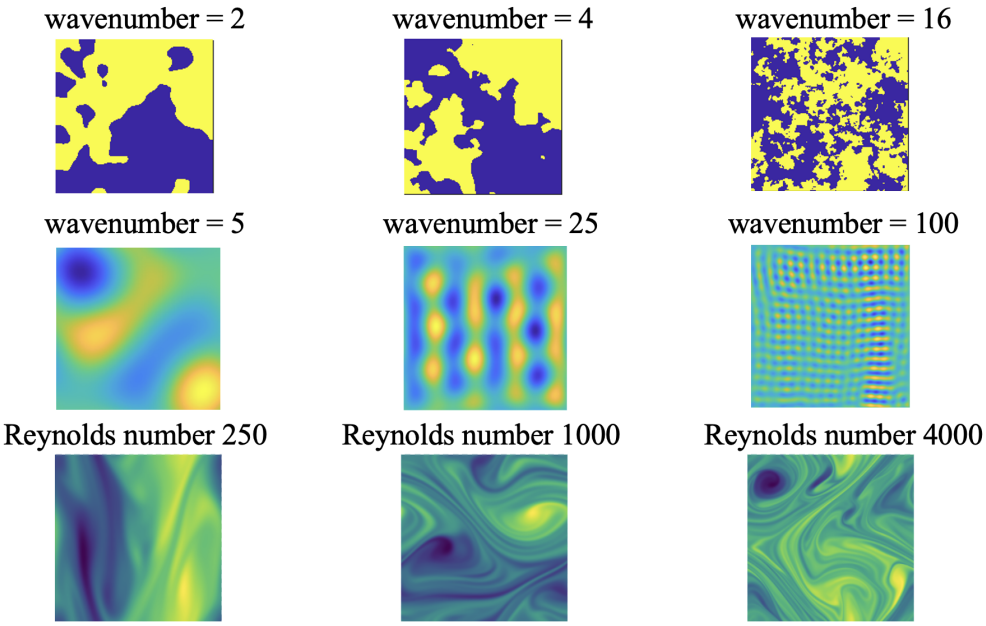


Figure 7.1: Multi-scale PDE dataset: Continuum mechanics at different scales (kilometer- or millimeter-scale) can be formulated to a unit-scaled domain with corresponding scale parameters. Row 1: Darcy Flows, Row 2: Helmholtz Equation, Row 3: Navier Stokes equation. In this work, we aim to design a learning framework to capture the consistency across the scales.

the Fourier Neural Operator (FNO) (Li, N. Kovachki, et al., 2020a) stands out as one of the most efficient models. It learns dynamics on the frequency domain, which can be viewed as an efficient, resolution-invariant tokenization. Recent advances further improve the model with shared kernel (Guibas et al., 2021) and U-shape architectures (Rahman, Ross, and Azizzadenesheli, 2022). Given promising results, one of the major challenges of scientific machine learning has been the lack of high-quality training data.

Self-supervised learning for PDEs. To overcome the limitation of data, many self-supervised learning techniques have been studied. Especially, the AI for Science community has investigated building-in physics knowledge to the models via equation loss (Li, Zheng, et al., 2021) and symmetry augmentation (R. Wang, Walters, and Yu, 2020; Brandstetter, Welling, and Worrall, 2022). For two-dimensional PDEs, the symmetry groups include translation, rotation, Galilean boost, and scaling. Among them, scale symmetry has been the least effective in improving performance (Brandstetter, Welling, and Worrall, 2022; Mialon et al., 2023). Our hypothesis is that previous formulations of scale-symmetry are defined as positional

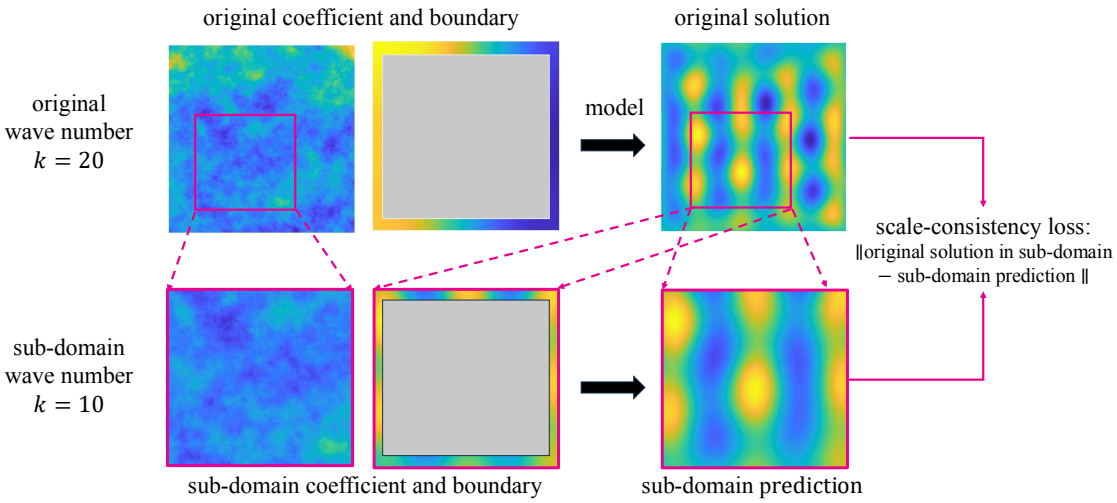


Figure 7.2: Scale-consistency loss via sub-domain sampling and re-scaling: given data instance of input coefficient, boundary, scale parameter, and solution we restrict them to a sub-domain to obtain new data instance, which is rescaled to unit length according to the ratio of subdomain size. The scale-consistency loss is defined as the discrepancy between the global and sub-domain prediction.

encoding, which does not incorporate scaling parameters and boundary conditions.

Multi-scale behavior in physics. Many natural phenomena exhibit multiscale behavior, i.e., interact across a wide range of scales. This is especially the case with solutions of partial differential equations (PDEs), which model various phenomena in science and engineering. For instance, the Navier-Stokes equation, a classical model describing fluid motion, applies to kilometer-scale problems such as weather forecasting (Pathak et al., 2022), meter-scale problems such as airfoils (Li, Nikola Borislavov Kovachki, et al., 2023), and millimeter-scale problems such as catheters (Zhou et al., 2023).

PDEs can be rescaled. While the physics at the kilometer and millimeter scales exhibit very different behaviors and frequency ranges, continuum mechanics can be universally reformulated in PDEs using scale parameters, such as the Reynolds number in the Navier-Stokes equation, as illustrated in figure 7.1.

Fact 1 (Rescaling of PDEs) *In general, a PDE R with coefficient function a and solution function u on domain Ω*

$$R\left(a(x), u(x)\right) = 0, \qquad (x \in \Omega) \tag{7.1}$$

can be rescaled to a new domain size Ω_λ with scaling λ,

$$R_\lambda\left(a(\lambda x), u(\lambda x)\right) = 0, \qquad (x \in \Omega_\lambda)$$

For example, in the Darcy flow, $R(a, u) = \nabla(a\nabla u)$.

Further, the solution operators on a given domain are consistent on the sub-domains. Thus, given a domain, the values of the PDE solution in a subdomain can be equivalently obtained by rescaling the subdomain to unit size but by choosing a different set of appropriate parameters (known as scale parameters) and boundary conditions in the PDE.

Our approach. Based on the above observation, we define the scale-consistency loss as the overall difference between the original solution of the PDE limited to the subdomain, and the one obtained from the modified PDE upon rescaling the subdomain to unit size, as shown in Figure 7.2. The ground-truth solution has a zero scale-consistency loss, or in other words, solution operators of PDEs are scale consistent.

We apply scale-consistency loss as a data augmentation procedure during the training of neural operators for solving PDEs. We address the challenging task of modeling PDEs that exhibit dramatically different behaviors across scales. Our new dataset consists of four PDEs at different scales: the Darcy flow with varying coefficients, the Burgers' equation with viscosity ranging from $1/100$ to $1/1000$, the Helmholtz equation with wavenumbers spanning 1 to 100, and the Navier-Stokes equation with Reynolds numbers from 250 to 10000, as shown in Figure 7.1. To evaluate generalization capabilities, we train models at specific scales and test their performance across different scales. In particularly challenging cases, such as the Helmholtz equation, different wavenumbers result in entirely distinct frequency ranges, causing all baseline models to fail at generalization. However, by incorporating our scale-consistency loss, the model successfully achieves zero-shot extrapolation to previously unseen scales, i.e., PDEs with scale parameters not available during training. Our main contributions are as follows.

- We propose a data augmentation scheme based on scale-consistency loss that creates data instances with various scales via sub- and super-sampling. For time-dependent problems, we sample in the space-time domain.

- We show a theorem (7.3.1) for elliptic PDEs that, under mild assumptions, low scale-consistency loss guarantees recovery of the underlying solution operator.
- We design a scale-informed neural operator that takes the scale parameter as input with weight-sharing parameterization and adaptive U-shape architecture to capture a wide range of scales.
- We propose a challenging multiscale dataset including the Burgers' equation, Darcy Flow, Helmholtz equation, and Navier-Stokes equation. The results show that the scale-consistency loss helps the scale-informed neural operator extrapolate to wider scales with a 38% error reduction on average compared to baseline FNO models at the cost of double runtime.

To capture a wide range of scales, we propose a new architecture named the scale-informed neural operator, as shown in Figure 7.3. We use the Fourier neural operator (FNO) (Li, N. Kovachki, et al., 2020a) as the backbone, as FNO naturally handles varying resolution by mapping inputs to the Fourier basis of unit domain size. We incorporate the scale parameter as an additional input and embed the scale features in the frequency space, helping the model capture different frequencies corresponding to different scale parameters. Inspired by (Guibas et al., 2021), we use a weight-sharing parameterization, where a single weight network is shared across all frequency modes. Additionally, it employs a multi-band U-shaped architecture similar to (Rahman, Ross, and Azizzadenesheli, 2022) that optimizes channel dimensions, using larger dimensions for lower frequency bands and smaller dimensions for higher frequency bands.

7.2 Related Work

Neural operator and foundation models. Data-driven models have become a common methodology to complement or augment numerical solvers for physical simulation (H. Wang et al., 2023). However, existing data-driven models are typically targeted to a single input variable, such as the coefficient function or initial condition, while other parameters remain fixed, including the domain size, boundary condition, and forcing term (Takamoto et al., 2022). Recently, foundation models have been proposed to capture various datasets under a wide range of conditions, or even multiple families of PDEs (Subramanian et al., 2024; McCabe et al., 2023; Hao et al., 2024; Shen, Marwah, and Talwalkar, 2024; Rahman, George, et al.,

2024). However, they do not explicitly capture relationships across a wide range of scales seen in physical systems. It is challenging for standard neural networks to capture different scales. In general, separate neural network models are trained for capturing each scale, making it cumbersome to couple them together and impose constraints across scales.

Symmetry. Scaling symmetry has been explored as a data augmentation technique in several works (R. Wang, Walters, and Yu, 2020; Brandstetter, Welling, and Worrall, 2022; Mialon et al., 2023). In dynamical systems, this symmetry represents a fundamental relationship between spatial coordinates, time evolution, and field magnitudes. However, both (Brandstetter, Welling, and Worrall, 2022) and (Mialon et al., 2023) reported limited effectiveness of this approach. This limitation may stem from two key challenges: first, continuous scaling symmetry becomes ill-defined on periodic domains without boundaries (Brandstetter, Welling, and Worrall, 2022), and second, scaling velocity magnitudes disrupts the natural range of the input space. This is particularly problematic in applications like weather forecasting, where velocity fields typically maintain consistent magnitude ranges. To address these limitations, we propose a generalized scaling consistenct framework that explicitly incorporates scaling parameters and boundary condition.

7.3 Scale-Consistency

Many PDEs possess symmetries, which are reflected by the fact that the equations remain invariant under transformations such as translation, rotation, or re-scaling. An example is the Darcy flow problem on a domain $\Omega \in \mathbb{R}^d$.

$$\begin{cases} -\nabla \cdot (a(x)\nabla u(x)) = 0, & (x \in \Omega), \quad (7.2a) \\ u(x) = g(x), & (x \in \partial\Omega). \quad (7.2b) \end{cases}$$

The associated solution operator $\mathcal{G}$ is defined as a mapping

$$(a(x), g(x)) \mapsto \mathcal{G}(a, g)(x) := u(x).$$

Scale symmetry and scale consistency

Re-scale symmetry. Let $\mathcal{T}_\lambda$ be the re-scaling operator with $\lambda \in \mathbb{R}^+$ defined by $(\mathcal{T}_\lambda a)(x) := a(\lambda x)$ (or more generally with translation $(\mathcal{T}_\lambda a)(x) = a(\lambda x + b)$ with $b \in \mathbb{R}^d$). In the absence of boundary conditions, the scale symmetry implies an equivariance property of $\mathcal{G}$:

$$\mathcal{G}(\mathcal{T}_\lambda a, \ldots) = \mathcal{T}_\lambda \mathcal{G}(a, \ldots).$$

The boundary condition (or simply the fact that the PDE is defined on a bounded domain Ω) breaks the scale symmetry; if $u : \Omega \to \mathbb{R}$ is defined on the domain Ω, then $\mathcal{T}_\lambda u$ is defined on the rescaled domain $\Omega_\lambda = \{\lambda^{-1}x | x \in \Omega\}$, and we are generally lacking information about the boundary condition of the re-scaled domain $\partial\Omega_\lambda$. Thus, the presence of boundaries in most problems of practical interest makes it difficult to leverage the underlying symmetry properties of the equations in a straightforward way.

Nevertheless, under some conditions on the domain Ω (e.g. $\Omega = [0,1]^d$ is a cube), the formal scale symmetry of the solution operator of (7.2) implies that if $u(x)$ solves (7.2) with coefficient field $a(x)$ and with boundary condition $g(x)$, then the rescaled function $u_\lambda(x) = \mathcal{T}_\lambda u(x) = u(\lambda x)$, solves

$$\begin{cases} -\nabla \cdot (a_\lambda(x)\nabla u_\lambda(x)) = 0, & (x \in \Omega_\lambda), \\ u_\lambda(x) = \mathcal{T}_\lambda u(x), & (x \in \partial\Omega_\lambda). \end{cases}$$

i.e. $u_\lambda(x)$ is a solution of the Darcy flow problem on domain Ω_λ, with coefficient field $a_\lambda = \mathcal{T}_\lambda a$, and boundary condition $(\mathcal{T}_\lambda u)|_{\partial\Omega_\lambda}$. Another operation we can perform is the restriction from Ω_λ to Ω when $\lambda \leq 1$. Intuitively, this condition expresses the fact that the solution operator of (7.2) is **scale-consistent**: The solution on a smaller subdomain $\Omega \subset \Omega_\lambda$ can either be obtained

leftmirgin=0.5cm by solving the PDE over the entire domain Ω_λ and then restricting the solution u to the smaller domain $u|_\Omega$.

leftmiirgiin=0.5cm by solving the PDE directly on the subdomain Ω, and imposing consistent boundary condition $u|_{\partial\Omega}$.

Combining the scale symmetry with restriction, we obtain a new equation (7.2) corresponding to the sub-domain of the original equation (7.3).

$$\begin{cases} -\nabla \cdot (a_\lambda(x)\nabla u_\lambda(x)) = 0, & (x \in \Omega), & \text{(7.3a)} \\ u_\lambda(x) = \mathcal{T}_\lambda u(x), & (x \in \partial\Omega). & \text{(7.3b)} \end{cases}$$

By uniqueness of the equation, the solution of (7.3) must be consistent with the original solution in (7.2).

Lemma 7.3.1 (Scale-consistency (solution function)) *If a function u satisfies equation* (7.2)*, then $u_\lambda = \mathcal{T}_\lambda u$ is the unique solution of equation* (7.3)*.*

Therefore, we obtain the following identity in terms of the solution operator $\mathcal{G}$: let $\lambda \leq 1$

$$\begin{aligned}[\mathcal{T}_\lambda \mathcal{G}(a,g)]|_\Omega &= \mathcal{G}([\mathcal{T}_\lambda a]|_\Omega, [\mathcal{T}_\lambda u]|_{\partial\Omega}) \\ &\equiv \mathcal{G}([\mathcal{T}_\lambda a]|_\Omega, [\mathcal{T}_\lambda \mathcal{G}(a,g)]|_{\partial\Omega}).\end{aligned} \tag{7.4}$$

For the solution operator, this identity holds for arbitrary inputs $a(x)$ and $g(x)$. The scale-consistency (7.4) can be used as a loss to train solution operators. Informally, if an operator satisfies (7.4), then it must be the target solution operator.

Theorem 7.3.1 (Scale-consistency (solution operator)) *If an operator Ψ satisfies the scale-consistency* (7.4) *and it matches the ground truth solution operator $\mathcal{G}$ on nearly constant coefficient functions, then $\Psi \equiv \mathcal{G}$.*

Scale-consistency loss. The first way to impose such a constraint is by introducing a loss of the form

$$L(a,g) = \|\mathcal{T}_\lambda \Psi(a,g) - \Psi(\mathcal{T}_\lambda a, \mathcal{T}_\lambda \Psi(a,g)|_{\partial\Omega})\|. \tag{7.5}$$

Note that this is an self-supervised loss term that doesn't require access to labeled data $u = \mathcal{G}(a,g)$. It only requires producing input function samples (a,g). When solution data u is available, the scale-consistency loss simplifies to

$$L(a,g) = \|\mathcal{T}_\lambda u - \Psi(\mathcal{T}_\lambda a, \mathcal{T}_\lambda u|_{\partial\Omega}))\|. \tag{7.6}$$

Infinitesimal scale-consistency. Another way to impose this constraint is by taking the λ-derivative of (7.4), leading to:

$$\partial_\lambda \mathcal{T}_\lambda \mathcal{G}(a,g) = \partial_\lambda \left[\mathcal{G}(\mathcal{T}_\lambda a, \mathcal{T}_\lambda \mathcal{G}(a,g)|_{\partial\Omega})\right].$$

We note that if $a(x)$ is a function, then the derivative $\partial_\lambda \mathcal{T}_\lambda a$ evaluated at $\lambda = 1$, is given by

$$\partial_\lambda \mathcal{T}_\lambda a|_{\lambda=1} = [\partial_\lambda a(\lambda x)]_{\lambda=1} = x \cdot \nabla a(x),$$

i.e., a radial spatial derivative of a. Substitution of this identity, and noting that $\mathcal{T}_{\lambda=1} a = a$ and $\mathcal{T}_{\lambda=1}\mathcal{G}(a,g)|_{\partial\Omega} = g$, implies that

$$\begin{aligned}&x \cdot \nabla_x [\mathcal{G}(a,g)](x) \\ &= \left\langle \frac{\delta \mathcal{G}(a,g)}{\delta a}, x \cdot \nabla_x a \right\rangle + \left\langle \frac{\delta \mathcal{G}(a,g)}{\delta g}, x \cdot \nabla_x [\mathcal{G}(a,g)] \right\rangle\end{aligned}$$

We observe that while (7.4) is highly non-linear, the infinitesimal constraint is quadratic in $\mathcal{G}$.

Scale-dependent problem: extension beyond scale symmetry

The scale-consistency constraint can be written in greater generality, even if the underlying PDE has no scale symmetry. In this case, the domain could be an input to the operator, and the relevant scale-consistency would be

$$\mathcal{G}(a, g; \Omega)|_{\Omega'} = \mathcal{G}(a|_{\Omega'}, \mathcal{G}(a, g, \Omega)|_{\partial\Omega'}; \Omega'), \quad (\Omega' \subset \Omega).$$

In some cases, this is equivalent to scaling certain parameters in the PDE, as explained below.

Helmholtz equation. An example not satisfying scale symmetry is the Helmholtz equation,

$$-\nabla \cdot (a(x)\nabla u(x)) + k^2 u(x) = f(x). \tag{7.7}$$

In this case, a rescaling of the spatial variable corresponds to a rescaling of the frequency k^2, i.e. $u_\lambda(x) = u(\lambda x)$ solves $-\nabla \cdot (a_\lambda(x)\nabla u_\lambda(x)) + \lambda^{-2}k^2 u_\lambda(x) = \lambda^{-2} f(\lambda x)$, or

$$-\nabla \cdot (a_\lambda(x)\nabla u_\lambda(x)) + k_\lambda^2 u_\lambda(x) = f_\lambda(x),$$

with $k_\lambda := \lambda^{-1}k$, $f_\lambda(x) := \lambda^{-2} f(\lambda x)$. Thus, the scale-consistency constraint involves the whole family of PDEs, $\Delta u + k^2 u = f$, for $k > 0$, with the transform on parameter $\mathcal{T}_\lambda(k) = \lambda k$.

Time-dependent problem: rescale in space-time domain

For time-dependent problems, in general, we could view the time dimension as another spatial dimension, and rescale both the spatial and temporal dimensions.

Navier-Stokes equation. Another example is the two-dimensional incompressible Navier-Stokes equation. In the velocity form, without forcing,

$$\partial_t u(x,t) + u(x,t) \cdot \nabla u(x,t) = \frac{1}{\rho}\nabla p(x,t) + \nu \Delta u(x,t),$$

The scaling is by $u_\lambda(x,t) = u(\lambda x, \lambda t)$, $p_\lambda(x,t) = p(\lambda x, \lambda t)$, and $\nu_\lambda := \lambda^{-1}\nu$. In the vorticity formulation where $\omega = \mathrm{curl}(u)$, we do not need to rescale the time.

$$\partial_t \omega(x,t) + u(x,t) \cdot \nabla \omega(x,t) = \nu \Delta \omega(x,t), \tag{7.8}$$

Rescaling the spatial variable x corresponds to rescaling the viscosity ν; $\omega_\lambda(x,t) = \omega(\lambda x, t)$ and $u_\lambda(x,t) = \lambda^{-1} u(\lambda x, t)$ solves

$$\partial_t \omega_\lambda(x,t) + u_\lambda(x,t) \cdot \nabla \omega_\lambda(x,t) = \nu_\lambda \Delta \omega_\lambda(x,t),$$

where $\nu_\lambda := \lambda^{-2}\nu$, here the coefficient λ in front of the term $(u_\lambda(x,t)\cdot\nabla\omega_\lambda(x,t))$ is absorbed by u_λ.

Main Algorithms

Remark: neural operator automatically rescales input to unit length. For standard neural networks such as convolution neural networks, re-scaling $\mathcal{T}$ needs to be implemented as interpolation. However, in the design of neural operators such as FNO, the domain size is implicitly re-scaled to unit length, where the Fourier basis is defined with length $[0,1]$. Given $\mathcal{T}_\lambda f$ defined on domain $[0,\lambda]$, Fourier neural operator Ψ automatically rescales it to unit length,

$$\Psi(\mathcal{T}_\lambda f,\ldots) := \Psi(\mathcal{T}_{1/\lambda}\mathcal{T}_\lambda f,\ldots) = \Psi(f,\ldots).$$

where f is defined on unit size $[0,1]$. Therefore, the re-scaling $\mathcal{T}$ is omitted in the algorithm.

Sub-domain sampling. The sub-domain sampling algorithm is based on equation (7.6), where we use sub-sampling (i.e., restrict to sub-domain) to obtain instance with smaller scale $\lambda k < k$. Given the input and output data $\{(a,g,k),u\}$ defined on domain Ω, we truncate the domain into a smaller sub-domain $\hat{\Omega}$. The input and output restriction to the sub-domain, along with the re-scaled parameter, become a new data instance $\{(\hat{a},\hat{g},\hat{k}),\hat{u}\}$. We compute the consistency loss as the difference between the model evaluated on restricted input $\Psi(\hat{a},\hat{g},\hat{k})$ and the restricted output $\hat{u}$.

Algorithm 1 Sub-domain sampling

1: **Input:** data tuple of coefficient, boundary, scale parameter, and solution $\{(a,g,k),u\}$ on domain $\Omega=[0,1]^2$, model Ψ, and sampling rate $\lambda<1$.
2: Sample the sub-domain $\hat{\Omega}=[w,w+\lambda]\times[h,h+\lambda]$, where $w,h\sim\mathrm{Unif}[0,1-\lambda]$.
3: Define new instance $(\hat{a}=a|_{\hat{\Omega}},\hat{g}=u|_{\partial\hat{\Omega}},\hat{k}=\lambda k),\hat{u}=u|_{\hat{\Omega}}$.
4: **Output:** scale-consistency loss $\|\Psi(\hat{a},\hat{g},\hat{k})-\hat{u}\|$.

Super-domain sampling. The super-domain sampling algorithm is based on equation (7.5), where we sample new instances corresponding to larger scale $\lambda k > k$. Given the distributions μ for a and ν for g, we can sample new instance a,g with larger scale λk and apply Algorithm 1. Different from 1, we do not have the ground truth output u on the larger scale. Instead, we estimate using the model $u=\Psi(a,g,\lambda k)$.

Algorithm 2 Super-domain sampling

1: **Input:** distributions of inputs coefficient and boundary μ, ν, model Ψ, scale parameter k, and sampling rate $\lambda > 1$.
2: Sample new instances $a \sim \mu, g \sim \nu$.
3: Define new scale as λk.
4: Estimate the solution of new domain $u = \Psi(a, g, \lambda k)$.
5: Call Algorithm 1 with input $\{(a, g, \lambda k), u\}$ and scale $1/\lambda$.
6: **Output:** scale-consistency loss $\|\Psi(a|_{\hat{\Omega}}, \Psi(a, g, \lambda k)|_{\partial\hat{\Omega}}, k) - \Psi(a, g, \lambda k)|_{\hat{\Omega}}\|$.

7.4 Scale-Informed Neural Operator

The scale-informed neural operator is based on the FNO Li, N. Kovachki, et al., 2020a, where convolution is implemented as a pointwise multiplication in the Fourier space. Since FNO automatically rescales its input to unit length, we design a scale embedding in the Fourier space to inform the model of the scale parameter k. Furthermore, we design a U-shaped architecture to optimize the channel dimension.

Embed scale parameters in Fourier Space

In the previous FNO, the weight tensor R is defined as a $(M_1 \times \cdots \times M_d \times C_{in} \times C_{out})$-tensor, which is sufficient for lower-dimensional problems with fewer total modes M. For larger-scale problems, such as highly turbulent flows, the weight tensor R becomes prohibitively large. Therefore, we propose an implicit representation of the weight tensor similar to AFNO Guibas et al., 2021, where the complex weight R with the shape $(C_{in} \times C_{out})$ is shared across all modes $(M_1 \times \cdots \times M_d)$.

Different from AFNO, we further define the features of scale k and mode index ξ as input, so that the transform R can behave correspondingly with respect to different scales k and modes ξ. Let C be the embedding channel dimension; we define scale features as $h(k)_i = k^{i/(C-1)}$ for $i = 0, 1, \ldots, C-1$, which covers a wide range from $k^{0/(C-1)} = 1$ to $k^{(C-1)/(C-1)} = k$. The input $f_t(\xi) \in \mathbb{C}^{C_{in}}$ is first element-wise multiplied with the features of the scale parameter and wavenumber $h(k, \xi)$, and then multiplied with R, followed by a group normalization and a complex activation σ. The transform $\mathcal{K}$ can be viewed as a kernel function defined on the Fourier space:

$$(\mathcal{K} f_{t+1})(\xi) = \sigma\big(R(f_t(\xi) \odot h(k, \xi))\big). \tag{7.9}$$

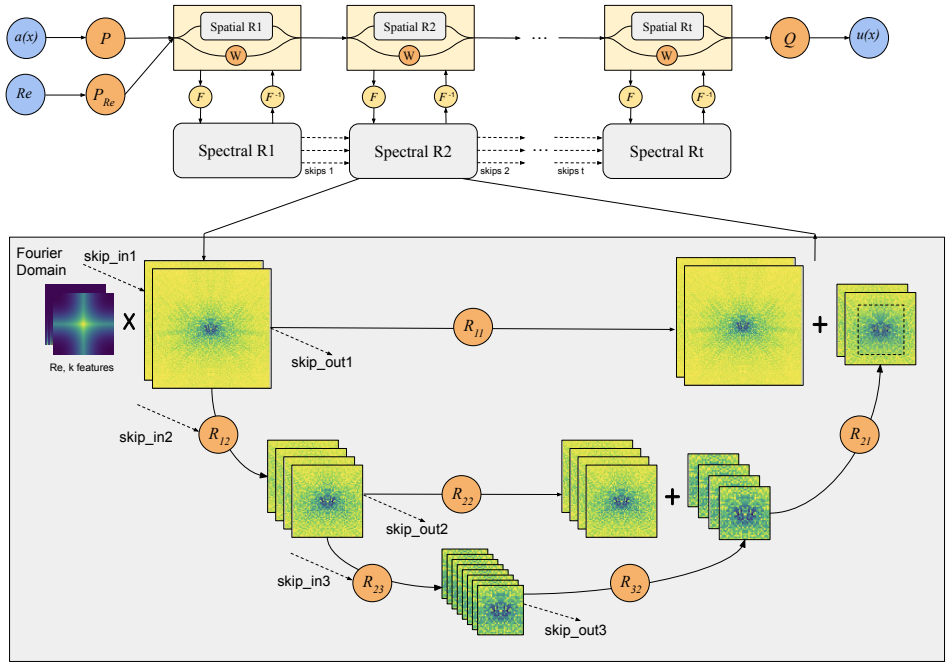


Figure 7.3: The scale-informed neural operator has a U-shape structure on the Fourier space. The scale parameter (such as Re) are embedded at each spectral layer. In the down block, the input tensors are truncated and lifted by complex layer R; in the up block, the tensors are projected and added to the inputs. Skip connections are added across the blocks. P is the encoder and Q is the decoder.

Multi-band Architecture

The Fourier signal usually follows an ordered structure, where the energy decays as the wavenumber increases. Therefore, previous methods such as FNO Li, N. Kovachki, et al., 2020a and SNO Fanaskov and Oseledets, 2022 choose to truncate to a fixed number of frequencies by omitting higher frequencies. Similar to previous works such as UNet Ronneberger, Fischer, and Brox, 2015 and UNO, we design a multi-band structure to gradually shrink the frequency bands, as shown in Figure 7.3. Different from UNO, which applies spectral convolutions at each down and up block, in this work, we define the U-shaped structure fully in the Fourier space. Given the initial channel dimension C, maximum input modes M, and a predefined number of levels L, we define C_l and M_l as $C_l = 2^l C$ and $M_l = 2^{-l} M$, where each block has shape $C_l^2 M_l^d$. For $d = 2$, $C_l^2 M_l^2 = C^2 M^2$, so each level has the same size. We define the first level using the weight-sharing formulation, where R_1 has the shape $(C_{in} \times C_{out})$, and higher levels in tensor formulation with $(M_l^d \times C_{in} \times C_{out})$.

Boundary condition

For boundary value problems, we take the boundary as an additional input. For a 1-dimensional boundary on a 2-dimensional square domain, we extend the boundary to 2D by repeating along the other dimension. For Dirichlet-type boundaries, it is known that the boundary is the restriction of the solution, and their magnitudes should be similar. Therefore, we define a normalization at the end of the model that multiplies the output by the magnitude of the boundary.

Darcy Flow (Scale)	2	3	4 (training)	8	16	
FNO	3.921	3.842	3.737	3.323	3.214	—
FNO+scale	**1.990**	**1.932**	**1.990**	**2.130**	**2.300**	—
UNet	6.638	6.981	6.011	5.527	6.361	—
UNet+scale	5.130	4.945	5.869	5.645	6.094	—
UNO	5.534	5.336	4.725	4.495	4.366	—
UNO+scale	3.009	2.753	3.087	4.602	4.600	—
Burgers' Equation (ν)	1/100	1/200	1/400 (training)	1/1000		
FNO	28.602	11.005	1.230	8.709	—	—
FNO+scale	27.799	10.008	1.908	9.442	—	—
SINO	24.914	10.027	1.174	8.363	—	—
SINO+scale	**5.926**	**1.720**	**0.957**	**4.575**	—	—
UNet	32.897	22.463	20.119	26.481	—	—
UNet+scale	30.137	22.815	25.138	30.747	—	—
UNO	28.581	10.963	1.235	8.624	—	—
UNO+scale	28.716	11.009	1.387	8.720	—	—
Helmholtz Equation (k)	1	2	5 (training)	10 (training)	25 (training)	50
FNO	136.847	131.200	4.285	12.575	21.060	107.186
FNO+scale	44.625	36.026	3.186	**11.924**	19.744	108.916
SINO-U	69.437	63.283	3.666	12.503	19.728	**102.980**
SINO-U+scale	**8.960**	**6.960**	**3.081**	12.490	**19.001**	112.940
UNet	164.945	156.775	48.341	51.028	21.189	112.914
UNet+scale	51.441	64.313	63.827	52.731	53.541	104.477
UNO	120.742	101.478	9.350	16.172	32.280	118.570
UNO+scale	125.742	91.541	10.821	19.605	36.017	117.776
Navier-Stokes, auto-reg. (Re)	250	500	1000 (training)	2000	4000	10000
FNO	0.447	0.750	1.015	3.108	7.374	18.295
FNO+scale	**0.302**	0.531	**0.743**	2.446	6.137	17.127
SINO-U	0.695	0.782	0.976	2.466	4.793	13.772
SINO-U + scale	0.357	**0.514**	0.953	2.186	**4.289**	**11.483**
UNet	4.156	2.706	0.809	**2.096**	10.027	22.284
UNet+scale	1.086	1.753	13.802	15.427	16.442	28.297
UNO	4.228	3.021	4.147	8.316	16.728	33.221
UNO+scale	4.005	2.661	3.458	6.941	14.785	30.663

Table 7.1: Comparison of FNO, UNet, UNO, and SINO with and without scale-consistency. Models are trained at certain scale and zero-shot test across others. Overall, scale-consistency helps each model extrapolate to unseen scales. Errors are in relative-L2 (%). The Darcy Flow is scale-invariant so the SINO does not apply.

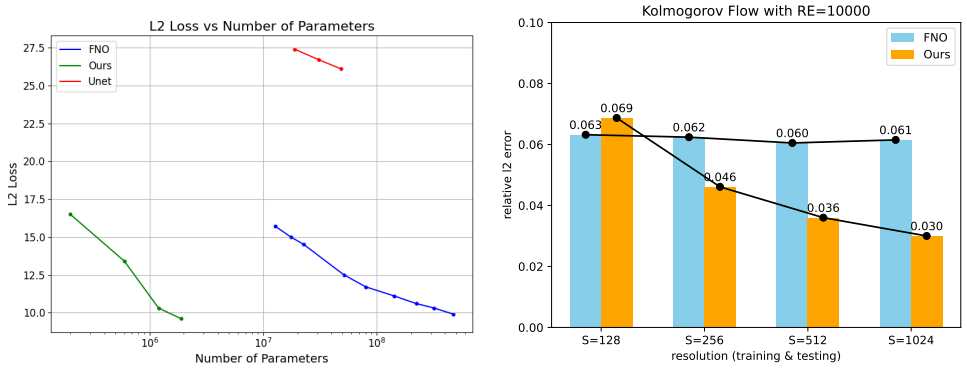


Figure 7.4: Ablation Studies of Scale Consistency. left: Cost-Accuracy: we train and test each model at various sizes on Kolmogorov Flow with RE=5000. Our model (u-shape) converges faster than baseline models. Further, the model (shared) achieves comparative accuracy with 1/10 of the parameters. right: discretization convergence: the proposed model does not truncate to a fixed bandwidth. As the training resolution increases, the model's error converges while the baseline FNO remains the same.

7.5 Experiments

We generated datasets for the Darcy Flow, Helmholtz equation, and Navier-Stokes equation, each spanning a wide range of scales. For each test case, we trained the models on a narrow range of scales and compared the performance with and without self-consistency augmentation. All experiments were run on Nvidia A100 (80GB, 40GB) or P100 (16GB) GPUs. The error metric is relative L2 error. The results show that self-consistency augmentation helps the model generalize better to unseen scales.

Self-consistency loss

In the first part, we compare FNO, UNet, and our models, with and without the self-consistency loss. For Darcy and Helmholtz equations, where the input distribution is given as a Gaussian random field, we apply both sub-sampling 1 and super-sampling 2. For the Navier-Stokes equation, the input distribution is unknown, so we only apply sub-sampling.

Darcy Flow. We considered the Darcy Flow (7.2) with a non-zero Dirichlet boundary. The input coefficient is sampled at different scale. The resolutions were $s = 64, 96, 128, 256, 512$, respectively. We train 1024 instances for training and 128 for testing. We used $\sigma = 1$ for training. Since Darcy has no scale parameters, we used FNO with and without scale-consistency. As shown in Table 7.1, FNO with

scale-consistency reduced the error by half compared to the baseline.

Helmholtz Equation. We considered the Helmholtz equation (7.7) with a non-zero Dirichlet boundary. The input coefficients a, g were sampled from a fixed Gaussian random field, with varying wavenumbers $k = 1, 2, 5, 10, 25, 50, 100$. The resolutions were $64, 64, 64, 128, 256, 512, 1024$, respectively. We train 1024 instances for training and 128 for testing. We used $k = 5, 10, 25$ for training. The scale-informed neural operator with scale-consistency reduced the error by half compared to the baseline FNO on smaller wavenumbers $k = 1, 2$, but neither model captured larger scales $k = 50, 100$, since Helmholtz equation has very different behaviors on larger scales.

Burgers' Equation. We considered the Burgers' equation. Given the initial condition and time-dependent boundary condition as input, the model predicts the solution over the next time interval. We train the FNO model and the multi-scale neural operator model (ours) with and without scale-consistency loss. The scale-consistent loss is across both the spatial and temporal domain. The models are trained on viscosity $= 1/400$ and tested on viscosities $\nu = 1/100, 1/200, 1/400, 1/1000$. The multi-scale neural operator with scale-consistency reduced the error up to 5x compared to the baseline FNO on unseen viscosity.

Navier-Stokes Equation (autoregressive). We considered the Navier-Stokes equation (7.8) defined on sub-domain similar to applications in climate. The input is the vorticity field of the previous ten time frames ω_0. We considered Reynolds numbers ranging from $Re = 250, 500, 1000, 2000, 4000, 10000$. The resolutions were $32, 64, 128, 128, 256, 512$, respectively. We train 50 trajectories for training and 5 (per each Re) for testing, where each trajectory consists of 300 time steps, with $dt = 0.1$. We used $Re = 1000$ for training. The multi-scale multi-band neural operator with scale-consistency reduced the error by 1/4 compared to the baseline UNet on unseen $Re = 250, 500, 4000, 10000$.

Navier-Stokes Equation (space-time, 2+1 dimensional). We also considered the spatiotemporal modeling for the Navier Stokes equation, velocity formulation. Similar to the autoregressive setting above, we considered Reynolds numbers ranging from $Re = 250, 500, 1000, 2000, 4000, 10000$. For continuous-time modeling, we use $dt = 1/256$ and are given input of the history, consisting of 24 frames, to predict the next 24 frames. We observe significant improvements with scale embedding and spatiotemporal cropping for out-of-distribution Reynolds numbers. Improvements are highlighted in Table 7.1.

Scale	2	3	4	8	16
No aug.	4.143	4.193	4.036	3.552	3.352
Rot.	3.101	2.944	2.953	2.797	2.872
Ref.	2.821	2.701	2.597	2.616	2.684
Rot.+Ref.	2.713	2.469	2.450	2.461	2.582
Scale (ours)	1.918	2.075	2.035	2.159	**2.237**
All (ours)	**1.903**	**1.816**	**1.910**	**2.095**	2.309

Table 7.2: Comparison of scale-consistency with existing symmetries for data augmentation, in relative-L2 error (%). We train FNO on Darcy flow at $scale = 4$ and zero-shot test at other scales.

Comparison with symmetry-based augmentation. On the Darcy flow, we compare the scale-consistency augmentation with existing symmetry-based augmentation as used in (R. Wang, Walters, and Yu, 2020; Brandstetter, Welling, and Worrall, 2022). As shown in Table 7.2, scale-consistency augmentation leads to better generalization compared to rotation plus reflection. Furthermore, scale-consistency works seamlessly with rotation and reflection. The best result is achieved by combining the three augmentation methods together.

Ablation studies on the model architecture

Embed scale parameter in the frequency space. We conducted several ablation studies on the proposed model architecture along with scale-consistency loss. We test the scale embedding and positional embedding on the frequency space (7.9) with Burgers' equation, Helmholtz equation, and Navier-Stokes equation. As shown in Table 7.3, the embedding in general improves the performance. We sometimes find the scale parameter is unnecessary in the Navier-Stokes equation when it can be inferred from the history of trajectory.

U-shape structure and shared kernel. We further conducted ablation studies on the U-shape structure model in the standard supervised learning setting on periodic Navier-Stokes equation with fixed scales $Re = 5000$ (with forcing) and $Re = 10000$ (zero forcing) as in (Li, Liu-Schiaffini, et al., 2022). For baselines, we consider FNO (Li, N. Kovachki, et al., 2020a), UNet (Ronneberger, Fischer, and Brox, 2015), FNO-UNet (Gupta and Brandstetter, 2022), and UNO (Rahman, Ross, and Azizzadenesheli, 2022). The results show that our model achieves a smaller error rate with one-tenth of the parameters compared to the previous FNO at the cost of longer runtime, as shown in Figure 7.4 (left). Since the model does not truncate

Burgers' equation							
Scale	**Freq. Emb.**	ν = 1/100	ν = 1/200	ν = 1/400	ν = 1/800		
No	No	28.531	10.756	1.087	8.889	—	—
No	Yes	28.660	10.832	**0.916**	8.725	—	—
Yes	No	10.731	2.540	1.055	5.477	—	—
Yes	Yes	**6.334**	**1.887**	1.042	**4.636**	—	—
Helmholtz equation							
Scale	**Freq. Emb.**	k = 1	k = 2	k = 5	k = 10	k = 25	k = 50
No	No	17.914	5.963	3.537	11.042	16.338	106.597
No	Yes	15.642	6.384	3.441	10.294	14.056	102.015
Yes	No	16.741	4.822	2.914	10.631	12.989	103.151
Yes	Yes	**9.438**	**4.980**	**2.921**	**9.874**	**11.574**	**93.938**

Table 7.3: Ablation for scale-informed neural operator on different equations in relative-L2 error (%). For Burgers' equation, we train on viscosity $\nu = 1/400$ and zero-shot test on other scales. For Helmholtz equation, we train on wavenumber $k = 5, 10, 25$.

Model	**Scale Informed**	**Freq. Emb.**	**Aug.**	**size min**	**Re=250 256**	**Re=500 256**	**Re=1000 512**	**Re=2000 512**	**Re=4000 1024**	**Re=10000 1024**
2+1 dim FNO	No	No	OFF	N/A	2.040	2.901	4.460	8.573	12.081	19.554
	No	Yes	OFF	N/A	1.727	2.632	4.051	8.158	11.603	18.847
	Yes	No	OFF	N/A	1.937	2.779	4.194	8.319	11.889	19.588
	Yes	Yes	OFF	N/A	1.756	2.551	4.003	8.029	11.274	18.551
2+1 dim FNO	Yes	Yes	ON	24	1.352	2.082	3.547	7.420	11.074	18.526
	Yes	Yes	ON	32	**1.342**	**2.016**	3.382	7.285	10.876	18.469
	Yes	Yes	ON	40	1.468	2.031	3.348	7.083	10.444	17.692
	Yes	Yes	ON	48	1.756	2.515	3.869	7.732	11.194	18.408
2+1 dim SINO	No	No	OFF	N/A	3.945	5.430	6.768	11.215	14.987	21.862
	No	Yes	OFF	N/A	3.586	4.348	2.827	6.307	15.211	23.422
	Yes	No	OFF	N/A	4.032	5.580	6.803	11.358	16.708	23.437
	Yes	Yes	OFF	N/A	2.125	2.661	**2.701**	**6.164**	11.917	19.405
2+1 dim SINO	No	No	ON	32	4.083	5.681	6.516	10.959	15.138	22.287
	No	Yes	ON	32	1.419	2.157	2.917	6.323	**9.880**	**17.095**
	Yes	No	ON	32	6.584	6.874	6.802	13.861	22.819	35.919
	Yes	Yes	ON	32	1.750	2.457	2.863	6.271	13.394	23.217

Table 7.4: Navier-Stokes equation trained on RE1000, zero-shot test on various RE (2+1 dimensional models). Values in percentage (%).

the maximum Fourier frequency, its accuracy improves as the resolution refines, as shown in Figure 7.4 (bottom right).

7.6 Discussion and Conclusion

In this paper, we consider the scale consistency for learning solution operators on PDEs across various scales. By leveraging the scale-consistency properties of PDEs and designing a scale-informed neural operator, we demonstrated the ability to model

a wide range of scales. Experimental results showed significant improvements in generalization to unseen scales, with better generalization errors compared to baseline models. This approach holds promise for improving the efficiency and generalizability of data-driven PDE solvers, reducing the need for extensive training data, and enabling the development of more flexible and foundational models for scientific and engineering applications.

While sub-sampling (Algorithm 1) is generally helpful, super-sampling (Algorithm 2) requires input distribution known to sample new instances. While the super-sampling works well for Darcy and Burgers, it is challenging to subsample from the attractor for the Navier-Stokes equation. As a potential future direction, it could be an interesting direction to combine with generative models (Lim et al., 2023) to sample virtual inputs.

References

Azizzadenesheli, Kamyar et al. (2024). "Neural operators for accelerating scientific simulations and design". In: *Nature Reviews Physics*, pp. 1–9.

Brandstetter, Johannes, Max Welling, and Daniel E Worrall (2022). "Lie point symmetry data augmentation for neural pde solvers". In: *International Conference on Machine Learning*. PMLR, pp. 2241–2256.

Fanaskov, Vladimir and Ivan Oseledets (2022). "Spectral neural operators". In: *arXiv preprint arXiv:2205.10573*.

Gopakumar, Vignesh et al. (2023). "Fourier Neural Operator for Plasma Modelling". In: *arXiv preprint arXiv:2302.06542*.

Guibas, John et al. (2021). "Adaptive fourier neural operators: Efficient token mixers for transformers". In: *arXiv preprint arXiv:2111.13587*.

Gupta, Jayesh K and Johannes Brandstetter (2022). "Towards multi-spatiotemporal-scale generalized pde modeling". In: *arXiv preprint arXiv:2209.15616*.

Hao, Zhongkai et al. (2024). "DPOT: Auto-Regressive Denoising Operator Transformer for Large-Scale PDE Pre-Training". In: *arXiv preprint arXiv:2403.03542*.

Kissas, Georgios et al. (2022). "Learning operators with coupled attention". In: *Journal of Machine Learning Research* 23.215, pp. 1–63.

Kovachki, Nikola B et al. (2023). "Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs." In: *J. Mach. Learn. Res.* 24.89, pp. 1–97.

Li, Zongyi, Nikola Kovachki, et al. (2020a). "Fourier Neural Operator for Parametric Partial Differential Equations". In: *arXiv preprint arXiv:2010.08895*.

Li, Zongyi, Nikola Kovachki, et al. (2020b). "Neural operator: Graph kernel network for partial differential equations". In: *arXiv preprint arXiv:2003.03485*.

Li, Zongyi, Nikola Borislavov Kovachki, et al. (2023). "Geometry-Informed Neural Operator for Large-Scale 3D PDEs". In: *arXiv preprint arXiv:2309.00583*.

Li, Zongyi, Miguel Liu-Schiaffini, et al. (2022). "Learning chaotic dynamics in dissipative systems". In: *Advances in Neural Information Processing Systems* 35, pp. 16768–16781.

Li, Zongyi, Hongkai Zheng, et al. (2021). "Physics-informed neural operator for learning partial differential equations". In: *ACM/JMS Journal of Data Science*.

Lim, Jae Hyun et al. (2023). "Score-based diffusion models in function space". In: *arXiv preprint arXiv:2302.07400*.

Lu, Lu et al. (Mar. 2021). "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". In: *Nature Machine Intelligence* 3.3, pp. 218–229.

McCabe, Michael et al. (2023). "Multiple physics pretraining for physical surrogate models". In: *arXiv preprint arXiv:2310.02994*.

Mialon, Grégoire et al. (2023). "Self-supervised learning with lie symmetries for partial differential equations". In: *Advances in Neural Information Processing Systems* 36, pp. 28973–29004.

Pathak, Jaideep et al. (2022). "Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators". In: *arXiv preprint arXiv:2202.11214*.

Rahman, Md Ashiqur, Robert Joseph George, et al. (2024). "Pretraining Codomain Attention Neural Operators for Solving Multiphysics PDEs". In: *arXiv preprint arXiv:2403.12553*.

Rahman, Md Ashiqur, Zachary E Ross, and Kamyar Azizzadenesheli (2022). "U-no: U-shaped neural operators". In: *arXiv preprint arXiv:2204.11127*.

Raonic, Bogdan et al. (2023). "Convolutional neural operators". In: *ICLR 2023 Workshop on Physics for Machine Learning*.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer, pp. 234–241.

Shen, Junhong, Tanya Marwah, and Ameet Talwalkar (2024). "Ups: Towards foundation models for pde solving via cross-modal adaptation". In: *arXiv preprint arXiv:2403.07187*.

Subramanian, Shashank et al. (2024). "Towards foundation models for scientific machine learning: Characterizing scaling and transfer behavior". In: *Advances in Neural Information Processing Systems* 36.

Takamoto, Makoto et al. (2022). "PDEBench: An extensive benchmark for scientific machine learning". In: *Advances in Neural Information Processing Systems* 35, pp. 1596–1611.

Wang, Hanchen et al. (2023). "Scientific discovery in the age of artificial intelligence". In: *Nature* 620.7972, pp. 47–60.

Wang, Rui, Robin Walters, and Rose Yu (2020). "Incorporating symmetry into deep dynamics models for improved generalization". In: *arXiv preprint arXiv:2002.03061*.

Zhou, Tingtao et al. (2023). "AI-aided Geometric Design of Anti-infection Catheters". In: *arXiv preprint arXiv:2304.14554*.

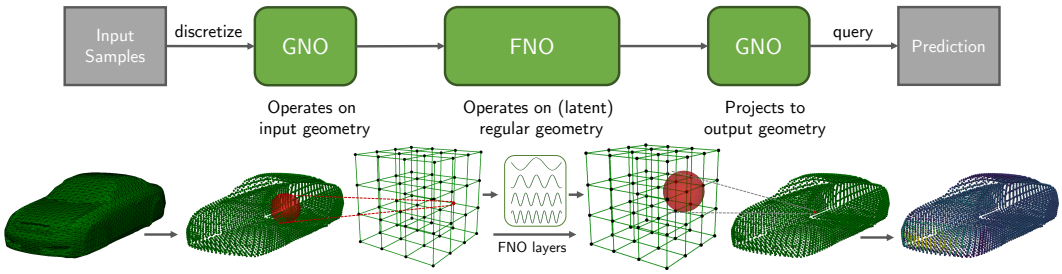
Chapter 8

GEOMETRY: NEURAL OPERATOR WITH LATENT SPACE

We propose the geometry-informed neural operator (GINO), a highly efficient approach for learning the solution operator of large-scale partial differential equations with varying geometries. GINO uses a signed distance function (SDF) and point-cloud representations of the input shape and neural operators based on graph and Fourier architectures to learn the solution operator. The graph neural operator handles irregular grids and transforms them into and from regular latent grids on which Fourier neural operator can be efficiently applied. GINO is discretization-convergent, meaning the trained model can be applied to arbitrary discretizations of the continuous domain and it converges to the continuum operator as the discretization is refined. To empirically validate the performance of our method on large-scale simulation, we generate the industry-standard aerodynamics dataset of 3D vehicle geometries with Reynolds numbers as high as five million. For this large-scale 3D fluid simulation, numerical methods are expensive to compute surface pressure. We successfully trained GINO to predict the pressure on car surfaces using only five hundred data points. The cost-accuracy experiments show a $26,000\times$ speed-up compared to optimized GPU-based computational fluid dynamics (CFD) simulators on computing the drag coefficient. When tested on new combinations of geometries and boundary conditions (inlet velocities), GINO obtains a one-fourth reduction in error rate compared to deep neural network approaches.

8.1 Introduction

Computational sciences aim to understand natural phenomena and develop computational models to study the physical world around us. Many natural phenomena follow the first principles of physics and are often described as evolution on function spaces, governed by partial differential equations (PDE). Various numerical methods, including finite difference and finite element methods, have been developed as computational approaches for solving PDEs. However, these methods need to be run at very high resolutions to capture detailed physics, which are time-consuming and expensive, and often beyond the available computation capacity. For instance, in computational fluid dynamics (CFD), given a shape design, the goal is to solve the Navier-Stokes equation and estimate physical properties such as pressure and



The input geometries are irregular and change for each sample. These are discretized into point clouds and passed on to a GNO layer, which maps from the given geometry to a latent regular grid. The output of this GNO layer is concatenated with the SDF features and passed into an FNO model. The output from the FNO model is projected back onto the domain of the input geometry for each query point using another GNO layer. This is used to predict the target function (e.g., pressure), which is used to compute the loss that is optimized end-to-end for training.

Figure 8.1: The architecture of GINO

velocity. Finding the optimal shape design often requires solving thousands of trial shapes, each of which can take more than ten hours even with GPUs (Korzun et al., 2022).

To overcome these computational challenges, recent works propose deep learning-based methods, particularly neural operators (Zongyi Li, Kovachki, Azizzadenesheli, B. Liu, Bhattacharya, et al., 2020b), to speed up the simulation and inverse design. Neural operators generalize neural networks and learn operators, which are mappings between infinite-dimensional function spaces (Zongyi Li, Kovachki, Azizzadenesheli, B. Liu, Bhattacharya, et al., 2020b). Neural operators are discretization convergent and can approximate general operators (Kovachki et al., 2021). The input function to neural operators can be presented at any discretization, grid, resolution, or mesh, and the output function can be evaluated at any arbitrary point. Neural operators have shown promise in learning solution operators in partial differential equations (PDE) (Kovachki et al., 2021) with numerous applications in scientific computing, including weather forecasting (Pathak et al., 2022), carbon dioxide storage and reservoir engineering (Wen et al., 2023), with a tremendous speedup over traditional methods. Prior works on neural operators developed a series of principled neural operator architectures to tackle a variety of scientific computing applications. Among the neural operators, graph neural operators (GNO) (Zongyi Li, Kovachki, Azizzadenesheli, B. Liu, Bhattacharya, et al., 2020b), and Fourier neural operators (FNO) (Zongyi Li, Kovachki, Azizzadenesheli, B. Liu, Bhattacharya, et al., 2020a) have been popular in various applications.

GNO implements kernel integration with graph structures and is applicable to complex geometries and irregular grids. The kernel integration in GNO shares similarities with the message-passing implementation of graph neural networks (GNN) (P. Battaglia et al., 2016), which is also used in scientific computing (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020; Allen et al., 2022). However, the main difference is that GNO defines the graph connection in a ball defined on the physical space, while GNN typically assumes a fixed set of neighbors, e.g., k-nearest neighbors, see Figure 8.2. Such nearest-neighbor connectivity in GNN violates discretization convergence, and it degenerates into a pointwise operator at high resolutions, leading to a poor approximation of the ground-truth operator using GNN. In contrast, GNO adapts the graph based on points within a physical space, allowing for universal approximation of operators. However, one limitation of graph-based methods is the computational complexity when applied to problems with long-range global interactions. To overcome this, prior works propose using multi-pole methods or multi-level graphs (Zongyi Li, Kovachki, Azizzadenesheli, B. Liu, Stuart, et al., 2020; Lam et al., 2022) to help with global connectivity. However, they do not fully alleviate the problem since they require many such levels to capture global dependence, which still makes them expensive.

While GNO performs kernel integration in the physical space using graph operations, FNO leverages Fourier transform to represent the kernel integration in the spectral domain using Fourier modes. This architecture is applicable to general geometries and domains since the (continuous) Fourier transform can be defined on any domain. However, it becomes computationally efficient when applied to regular input grids since the continuous Fourier transform can then be efficiently approximated using discrete Fast Fourier transform (FFT) (Cooley and Tukey, 1965), giving FNO a significant quasi-linear computational complexity. However, FFT limits FNO to regular grids and cannot directly deal with complex geometries and irregular grids. A recent model, termed GeoFNO, learns a deformation from a given geometry to a latent regular grid (Zongyi Li, D. Z. Huang, et al., 2022) so that the FFT can be applied in the latent space. In order to transform the latent regular grid back to the irregular physical domain, discrete Fourier transform (DFT) on irregular grids is employed. However, DFT on irregular grids is more expensive than FFT, quadratic vs. quasi-linear, and does not approximate the Fourier transform in a discretization convergent manner. This is because, unlike in the regular setting, the points are not sampled at regular intervals, and therefore the integral does not take into account the underlying measure. Other attempts share a similar computational barrier as

shown in Table 8.1, which we discussed in Section 8.2.

In this paper, we consider learning the solution operator for large-scale PDEs, in particular, 3D CFD simulations. We propose the geometry-informed neural operator (GINO), a neural operator architecture for arbitrary geometries and mesh discretizations. It uses a signed distance function (SDF) to represent the geometry and composes GNO and FNO architectures together in a principled manner to exploit the strengths of both frameworks.

The GNO by itself can handle irregular grids through graphs but is able to operate only locally under a limited computational budget, while the FNO can capture global interactions, but requires a regular grid. By using GNO to transform the irregular grid into a regular one for the FNO block, we can get the best of both worlds, i.e., computational efficiency and accuracy of the approximation. Thus, this architecture tackles the issue of expensive global integration operations that were unaddressed in prior works, while maintaining discretization convergence.

Specifically, GINO has three main components, (i) **Geometry encoder**: multiple local kernel integration layers through GNO with graph operations, (ii) **Global model**: a sequence of FNO layers for global kernel integration, and (iii) **Geometry decoder**: the final kernel integral layers, as shown in Figure 8.1. The input to the GINO is the input surface (as a point cloud) along with the SDF, representing the distance of each 3D point to the surface. GINO is trained end-to-end to predict output (e.g., car surface pressure in our experiments), a function defined on the geometry surfaces.

Geometry encoder: the first component in the GINO architecture uses the surface (i.e., point cloud) and SDF features as inputs. The irregular grid representation of the surface is encoded through local kernel integration layers implemented with GNOs, consisting of local graphs that can handle different geometries and irregular grids. The encoded function is evaluated on a regular grid, which is concatenated with the SDF input evaluated on the same grid. **Global model:** the output of the first component is encoded on a regular grid, enabling efficient learning with an FNO using FFT. Our second component consists of multiple FNO layers for efficient global integration. In practice, we find that this step can be performed at a lower resolution without significantly impacting accuracy, giving a further computational advantage. **Geometry decoder:** the final component is composed of local GNO-based layers with graph operations, that decode the output of the FNO and project it back onto the desired geometry, making it possible to efficiently query the output

Model	Range	Complexity	Irreg. grid	Disc. conv.
GNN	local	$O(N\text{degree})$	Yes	No
CNN	local	$O(N)$	No	No
UNet	global	$O(N)$	No	No
Transformer	global	$O(N^2)$	Yes	Yes
GNO (kernel)	radius r	$O(N\text{degree})$	Yes	Yes
FNO (FFT)	global	$O(N \log N)$	No	Yes
GINO [**Ours**]	global	$O(N \log N + N\text{degree})$	Yes	Yes

N is the number of mesh points; d is the dimension of the domain and degree is the maximum degree of the graph. Even though GNO and transformer both work on irregular grids and are discretization convergent, they become too expensive on large-scale problems.

Table 8.1: Computational complexity of standard deep learning models

function on irregular meshes. The GNO layers in our framework are accelerated using our GPU-based hash-table implementation of neighborhood search for graph connectivity of meshes.

We validate our findings on two large-scale 3D CFD datasets. We generate our own large-scale industry-standard Ahmed's body geometries using GPU-based OpenFOAM (Jasak, Jemcov, Tukovic, et al., 2007), composed of 500+ car geometries with $O(10^5)$ mesh points on the surface and $O(10^7)$ mesh points in space. Each simulation takes 7-19 hours on 16 CPU cores and 2 Nvidia V100 GPUs. Further, we also study a lower resolution dataset with more realistic car shapes, viz., Shape-Net car geometries generated by Umetani and Bickel, 2018. GINO takes the point clouds and SDF features as the input and predicts the pressure fields on the surfaces of the vehicles. We perform a full cost-accuracy trade-off analysis. The result shows GINO is $26,000\times$ faster at computing the drag coefficients over the GPU-based OpenFOAM solver, while achieving 8.31% (Ahmed-body) and 7.29% (Shape-Net car) error rates on the full pressure field. Further, GINO is capable of zero-shot super-resolution, training with only one-eighth of the mesh points, and having a good accuracy when evaluated on the full mesh that is not seen during training.

8.2 Related Work

The study of neural operators and their extended applications in learning solution operators in PDE has been gaining momentum (Kovachki et al., 2021; Bhattacharya et al., 2021; Kissas et al., 2022; Rahman, Ross, and Azizzadenesheli, 2022). A method that stands out is FNO, which uses Fourier transform (Zongyi Li, Kovachki,

Azizzadenesheli, B. Liu, Bhattacharya, et al., 2020a). The FNO and its variations have proven to highly accelerate the simulations for large-scale flow problems, including weather forecasting (Pathak et al., 2022), seismology (Yang et al., 2021; Sun et al., 2022) and multi-phase flow (Wen et al., 2023). However, a challenge with the FNO is that its computation superiority is gained when applied on a regular grid, where the Fourier transform is approximated using FFT. Therefore, its reliance on FFT limits its use with irregular grids or complex geometries. There have been attempts to modify the FNO to work with these irregular structures, but scalability to large-scale 3D PDEs remains an issue. One such attempt is GeoFNO, which learns a coordinate transformation to map irregular inputs to a regular latent space (Zongyi Li, D. Z. Huang, et al., 2022). This method, while innovative, requires a geometric discrete Fourier transform, which is computationally demanding and lacks discretization insurance. To circumvent this, GINO limits the Fourier transform to a local GNO to improve efficiency. The locality is defined assuming the metrics of the physical space.

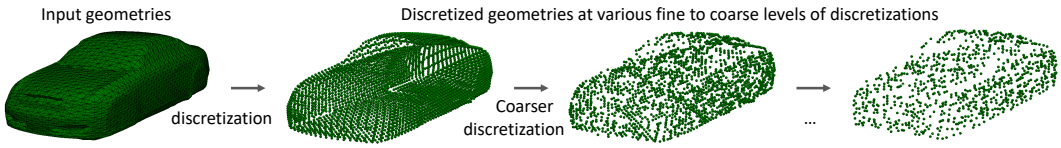
Additionally, the Non-Equispaced Fourier neural solvers (NFS) merge the FNO with non-equispaced interpolation layers, a method similar to global GNO (Lin et al., 2022). However, at the architecture level, their method replaces the integration of GNO with the summation of the nearest neighbor points on the graph. This step transitions this method to a neural network, failing to deliver a discretization convergent approach. The Domain-Agnostic Fourier Neural Operators (DAFNO) represents another attempt at improvement, applying an FNO to inputs where the geometry is represented as an indicator function (N. Liu, Jafarzadeh, and Yu, 2023). However, this method lacks a strategy for handling irregular point clouds. Simultaneously, researchers are exploring the combination of FNO with the attention mechanisms (Kovachki et al., 2021) for irregular meshes. This includes the Operator Transformer (OFormer) (Zijie Li, Meidani, and Farimani, 2022), Mesh-Independent Neural Operator (MINO) (Lee, n.d.), and the General Neural Operator Transformer (GNOT) (Hao et al., 2023). Besides, the Clifford neural layers (Brandstetter et al., 2022) use the Clifford algebra to compute multivectors, which provides Clifford-FNO implementations as an extension of FNO. The work (Wandel, Weinmann, and Klein, 2021) uses a physics-informed loss with U-Net for 3D channel flow simulation with several shapes. The work (Balu et al., 2021) innovatively proposes the use of multigrid training for neural networks that improves the convergence. Although these methods incorporate attention layers, which are special types of kernel integration Kovachki et al., 2021 with quadratic complexity, they face challenges when

scaling up for large-scale problems.

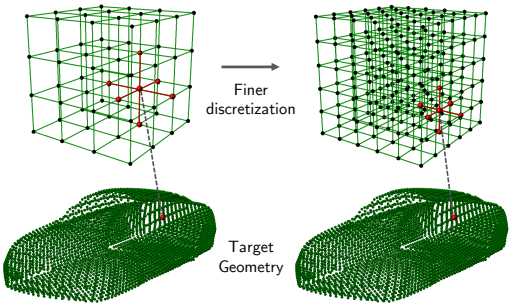
GNNs are incorporated in the prior attempts in physical simulations involving complex geometry, primarily due to the inherent flexibility of graph structures. Early research (P. Battaglia et al., 2016; Kipf and Welling, 2016; Hamilton, Ying, and Leskovec, 2017; P. W. Battaglia et al., 2018) laid the foundation for GNNs, demonstrating that physical entities, when represented as graph nodes, and their interactions, as edges, could predict the dynamics of various systems. The introduction of graph element networks (Alet et al., 2019) marked a significant development, being the first to apply GNNs to PDEs by discretizing the domain into elements. The similar idea has been explored in term of continuous 3D point cloud convolution (Y. Li et al., 2018; Hermosilla et al., 2018; W. Wu, Qi, and Fuxin, 2019). Another line of work, mesh graph networks (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020; Allen et al., 2022), further explored PDEs in the context of fluid and solid mechanics. Remelli et al., 2020; Durasov et al., 2021 train a Graph convolutional neural works on the ShapeNet car dataset for inverse design. However, GNN architectures' limitations hinder their use in operator learning for PDEs. GNNs connect each node to its nearest neighbors according to the graph's metrics, not the metrics of the physical domain. As the input function's discretization becomes finer, each node's nearest neighbors eventually converge to the same node, contradicting the expectation of improved model performance with finer discretization. Furthermore, GNNs' model behavior at the continuous function limit lacks a unique definition, failing the discretization convergence criterion. Consequently, as pointwise operators in function spaces at the continuous limit, GNNs struggle to approximate general operators between function spaces, Figure 8.2.

8.3 Problem setting

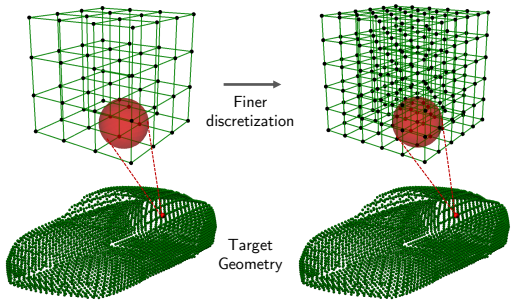
We are interested in learning the map from the geometry of a PDE to its solution. We will first give a general framework and then discuss the Navier-Stokes equation in CFD as an example. Let $D \subset \mathbb{R}^d$ be a Lipschitz domain and $\mathcal{A}$ a Banach space of real-valued functions on D. We consider the set of distance functions $\mathcal{T} \subset \mathcal{A}$ so that, for each function $T \in \mathcal{T}$, its zero set $S_T = \{x \in D : T(x) = 0\}$ defines a $(d-1)$-dimensional sub-manifold. We assume S_T is simply connected, closed, smooth, and that there exists $\epsilon > 0$ such that $B_\epsilon(x) \cap \partial D = \varnothing$ for every $x \in S_T$ and $T \in \mathcal{T}$. We denote by $Q_T \subset D$, the open volume enclosed by the sub-manifold S_T and assume that Q_T is a Lipschitz domain with $\partial Q_T = S_T$. We define the Lipschitz domain $\Omega_T := D \setminus \bar{Q}_T$ so that, $\partial\Omega_T = \partial D \cup S_T$. Let $\mathcal{L}$ denote a partial differential



(a) **An input geometry** (continuous function) is first discretized into a series of points by subsampling it. Note that in practice, the discretization can be highly irregular. A key challenge with several scientific computing applications is that we want a method that can work on arbitrary geometries, but also that is discretization convergent, meaning that the method converges to a desired solution operator as we make the discretization finer.



(b) **GNN** connects each point in the latent subspace (red) to its nearest neighbors in the original space (top). This is very discretization dependent, and as we increase the resolution (sample points more densely), the method becomes increasingly local and fails to capture context. In addition, the operator at the discretization limit is non-unique and depends on how the discretization is done.



(c) **GNO** instead connects each point in the latent subspace (red) to all its neighbors within an epsilon ball in the original space (top). This induces convergence to a continuum solution operator as we increase the resolution (sample points more densely). This means GNO converges to a unique operator as the discretization becomes finer and scales to large problems.

Figure 8.2: Comparison of GNN and GNO as the discretization becomes finer. GNN is discretization dependent, while GNO is discretization convergent.

operator and consider the problem

$$\begin{aligned} \mathcal{L}(u) &= f, \qquad \text{in } \Omega_T, \\ u &= g, \qquad \text{in } \partial\Omega_T, \end{aligned} \tag{8.1}$$

for some $f \in \mathcal{F}$, $g \in \mathcal{B}$ where $\mathcal{B}, \mathcal{F}$ denote Banach spaces of functions on $\mathbb{R}^d$ with the assumption that the evaluation functional is continuous in $\mathcal{B}$. We assume that $\mathcal{L}$ is such that, for any triplet (T, f, g), the PDE (8.1) has a unique solution $u \in \mathcal{U}_T$ where $\mathcal{U}_T$ denotes a Banach space of functions on Ω_T. Let $\mathcal{U}$ denote a Banach space of functions on D and let $\{E_T : \mathcal{U}_T \to \mathcal{U} : T \in \mathcal{T}\}$ be a family of extension operators which are linear and bounded. We define the mapping from the distance

function to the solution function

$$\Psi : \mathcal{T} \times \mathcal{F} \times \mathcal{B} \to \mathcal{U} \tag{8.2}$$

by $(T, f, g) \mapsto E_T(u)$ which is our operator of interest.

Navier-Stokes Equation. We illustrate the above abstract formulation with the following example. Let $D = (0,1)^d$ be the unit cube and let $\mathcal{A} = C(\bar{D})$. We take $\mathcal{T} \subset \mathcal{A}$ to be some subset such that the zero level set of every element defines a $(d-1)$-dimensional closed surface which can be realized as the graph of a Lipschitz function and that there exists $\epsilon > 0$ such that each surface is at least distance ϵ away from the boundary of D. We now consider the steady Naiver-Stokes equations,

$$\begin{aligned} -\nu\Delta v + (v \cdot \nabla)v + \nabla p &= f, && \text{in } \Omega_T, \\ \nabla \cdot v &= 0, && \text{in } \Omega_T, \\ v &= q, && \text{in } \partial D, \\ v &= 0, && \text{in } S_T, \end{aligned} \tag{8.3}$$

where $v : \Omega_T \to \mathbb{R}^d$ is the velocity, $p : \Omega_T \to \mathbb{R}$ is the pressure, ν is the viscosity, and $f, q : \mathbb{R}^d \to \mathbb{R}^d$ are the forcing and boundary functions. The condition that $v = 0$ in S_T is commonly known as a no slip boundary and is prevalent in many engineering applications. The function q, on the other hand, defines the inlet and outlet boundary conditions for the flow. We assume that $f \in H^{-1}(\mathbb{R}^d; \mathbb{R}^d)$ and $q \in C(\mathbb{R}^d; \mathbb{R}^d)$. We can then define our boundary function $g \in C(\mathbb{R}^d; \mathbb{R}^d)$ such that $g(x) = 0$ for any $x \in D$ with $\operatorname{dist}(x, \partial D) \geq \epsilon$ and $g(x) = q(x)$ for any $x \in D$ with, $\operatorname{dist}(x, \partial D) > \epsilon/2$ as well as any $x \notin D$. Continuity of g can be ensured by an appropriate extension for any $x \in D$ such that $\operatorname{dist}(x, \partial D) < \epsilon$ and $\operatorname{dist}(x, \partial D) \geq \epsilon/2$ Whitney, 1934. We define $u : \Omega_T \to \mathbb{R}^{d+1}$ by $u = (v, p)$ as the unique weak solution of (8.3) with $\mathcal{U}_T = H^1(\Omega_T; \mathbb{R}^d) \times L^2(\Omega_T)/\mathbb{R}$ Temam, 1984. We define $\mathcal{U} = H^1(D; \mathbb{R}^d) \times L^2(D)/\mathbb{R}$ and the family of extension operators $\{E_T : \mathcal{U}_T \to \mathcal{U}\}$ by $E_T(u) = \left(E_T^v(v), E_T^p(p)\right)$ where $E_T^v : H^1(\Omega_T; \mathbb{R}^d) \to H^1(D; \mathbb{R}^d)$ and $E_T^p : L^2(\Omega_T)/\mathbb{R} \to L^2(D)/\mathbb{R}$ are defined as the restriction onto D of the extension operators defined in Stein, 1970, Chapter 6, Theorem 5. This establishes the existence of the operator $\Psi : \mathcal{T} \times H^{-1}(\mathbb{R}^d; \mathbb{R}^d) \times C(\mathbb{R}^d; \mathbb{R}^d) \to H^1(D; \mathbb{R}^d) \times L^2(D)/\mathbb{R}$ mapping the geometry, forcing, and boundary condition to the (extended) solution of the steady Navier-Stokes equation (8.3). Homomorphic extensions of deformation-based operators have been shown in Cohen, Schwab, and Zech, 2018. We leave for future work studying the regularity properties of the presently defined operator.

8.4 Geometric-Informed Neural Operator

We propose a geometry-informed neural operator (GINO), a neural operator architecture for varying geometries and mesh regularities. GINO is a deep neural operator model consisting of three main components, (i) multiple local kernel integration layers, (ii) a sequence of FNO layers for global kernel integration which precedes (iii) the final kernel integral layers. Each layer of GINO follows the form of generic kernel integral of the form (8.5). Local integration is computed using graphs, while global integration is done in Fourier space.

Neural operator

A neural operator Ψ Kovachki et al., 2021 maps the input functions $a = (T, f, g)$ to the solution function u. The neural operator Ψ is composed of multiple layers of point-wise and integral operators,

$$\Psi = \mathbb{Q} \circ \mathcal{K}_L \circ \ldots \circ \mathcal{K}_1 \circ \P. \tag{8.4}$$

The first layer $\P$ is a pointwise operator parameterized by a neural network. It transforms the input function a into a higher-dimensional latent space $\P : a \mapsto v_0$. Similarly, the last layer acts as a projection layer, which is a pointwise operator $\mathbb{Q} : v_l \mapsto u$, parameterized by a neural network Q. The model consists of L layers of integral operators $\mathcal{K}_l : v_{l-1} \mapsto v_l$ in between.

$$v_l(x) = \int_D \kappa_l(x, y) v_{l-1}(y) \mathrm{d}y \tag{8.5}$$

where κ_l is a learnable kernel function. Non-linear activation functions are incorporated between each layer.

Graph operator block

To efficiently compute the integral in equation (8.5), we truncate the integral to a local ball at x with radius $r > 0$, as done in (Zongyi Li, Kovachki, Azizzadenesheli, B. Liu, Bhattacharya, et al., 2020b),

$$v_l(x) = \int_{B_r(x)} \kappa(x, y) v_{l-1}(y) \, \mathrm{d}y. \tag{8.6}$$

We discretize the space and use a Riemann sum to compute the integral. This process involves uniformly sampling the input mesh points and connecting them with a graph for efficient parallel computation. Specifically, for each point $x \in D$, we randomly sample points $\{y_1, \ldots, y_M\} \subset B_r(x)$ and approximate equation (8.6)

as

$$v_l(x) \approx \sum_{i=1}^{M} \kappa(x, y_i) v_{l-1}(y_i) \mu(y_i), \tag{8.7}$$

where μ denotes the Riemannian sum weights corresponding to the ambient space of $B_r(x)$. For a fixed input mesh of N points, the computational cost of equation (8.7) scales with the number of edges, denoted as $O(E) = O(MN)$. Here, the number of sampling points M is the degree of the graph. It can be either fixed to a constant sampling size, or scale with the area of the ball.

Encoder. Given an input point cloud $\{x_1^{\text{in}}, \ldots, x_N^{\text{in}}\} \subset S_T$, we employ a GNO-encoder to transform it to a function on a uniform latent grid $\{x_1^{\text{grid}}, \ldots, x_S^{\text{grid}}\} \subset D$. The encoder is computed as discretization of an integral operator $v_0(x^{\text{grid}}) \approx \sum_{i=1}^{M} \kappa(x^{\text{grid}}, y_i^{\text{in}}) \mu(y_i^{\text{in}})$ over ball $B_{r_{\text{in}}}(x^{\text{grid}})$. To inform the grid density, GINO computes Riemannian sum weights $\mu(y_i^{\text{in}})$. Further, we use Fourier features in the kernel Sitzmann et al., 2020. For simple geometries, this encoder can be omitted, see Section 8.5.

Decoder. Similarly, given a function defined on the uniform latent grid $\{x_1^{\text{grid}}, \ldots, x_S^{\text{grid}}\} \subset D$, we use a GNO-decoder to query arbitrary output points $\{x_1^{\text{out}}, \ldots, x_N^{\text{out}}\} \subset \Omega_T$. The output is evaluated as $u(x^{\text{out}}) \approx \sum_{i=1}^{M} \kappa(x^{\text{out}}, y_i^{\text{grid}}) v_l(y_i^{\text{grid}}) \mu(y_i^{\text{grid}})$ over ball $B_{r_{\text{out}}}(x^{\text{out}})$. Here, the Riemannian weight, $\mu(y_i^{\text{grid}}) = 1/S$ since we choose the latent space to be regular grid. Since the queries are independent, we divide the output points into small batches and run them in parallel, which enables us to use much larger models by saving memory.

Efficient graph construction. The graph construction requires finding neighbors to each node that are within a certain radius. The simplest solution is to compute all possible distances between neighbors, which requires $O(N^2)$ computation and memory. However, as the N gets larger, e.g., $10 \sim 100$ million, computation and memory become prohibitive even on modern GPUs. Instead, we use a hash grid-based implementation to efficiently prune candidates that are outside of a ℓ^∞-ball first and then compute the ℓ^2 distance between only the candidates that survive. This reduces the computational complexity to $O(Ndr^3)$ where d denotes unit density and r is the radius. This can be efficiently done using first creating a hash table of voxels with size r. Then, for each node, we go over all immediate neighbors to the current voxel that the current node falls into and compute the distance between all points

in these neighboring voxels. Specifically, we use the CUDA implementation from Open3D Zhou, Park, and Koltun, 2018. Then, using the neighbors, we compute the kernel integration using gather-scatter operations from torch-scatter Fey et al., 2023. Further, if the degree of the graph gets larger, we can add Nyström approximation by sampling nodes (Zongyi Li, Kovachki, Azizzadenesheli, B. Liu, Bhattacharya, et al., 2020b).

Fourier operator block

The geometry encoding v_0 and the geometry specifying map T, both evaluated on a regular grid discretizing D are passed to a FNO block. We describe the basic FNO block as first outlined in Zongyi Li, Kovachki, Azizzadenesheli, B. Liu, Bhattacharya, et al., 2020a. We will first define global convolution in the Fourier space and use it to build the full FNO operator block. To that end, we will work on the d-dimensional unit torus $\mathbb{T}^d$. We define an integral operator with kernel $\kappa \in L^2(\mathbb{T}^d; \mathbb{R}^{n\times m})$ as the mapping $C : L^2(\mathbb{T}^d; \mathbb{R}^m) \to L^2(\mathbb{T}^d; \mathbb{R}^n)$ given by

$$C(v) = \mathcal{F}^{-1}\big(\mathcal{F}(\kappa) \cdot \mathcal{F}(v)\big), \qquad \forall\, v \in L^2(\mathbb{T}^d; \mathbb{R}^m)$$

Here $\mathcal{F}, \mathcal{F}^{-1}$ are the Fourier transform and its inverse respectively, defined for L^2 by the appropriate limiting procedure. The Fourier transform of the function κ will be parameterized directly by some fixed number of Fourier modes, denoted $\alpha \in \mathbb{N}$. In particular, we assume

$$\kappa(x) = \sum_{\gamma\in I} c_\gamma \mathrm{e}^{i\langle \gamma, x\rangle}, \qquad \forall\, x \in \mathbb{T}^d$$

for some index set $I \subset \mathbb{Z}^d$ with $|I| = \alpha$ and coefficients $c_\gamma \in \mathbb{C}^{n\times m}$. Then we may view $\mathcal{F} : L^2(\mathbb{T}^d; \mathbb{R}^{n\times m}) \to \ell^2(\mathbb{Z}^d; \mathbb{C}^{n\times m})$ so that $\mathcal{F}(\kappa)(\gamma) = c_\gamma$ if $\gamma \in I$ and $\mathcal{F}(\kappa)(\gamma) = 0$ if $\gamma \notin I$. We directly learn the coefficients c_γ without ever having to evaluate κ in physical space. We then define the full operator block $\mathcal{K} : L^2(\mathbb{T}^d; \mathbb{R}^m) \to L^2(\mathbb{T}^d; \mathbb{R}^n)$ by

$$\mathcal{K}(v)(x) = \sigma\big(Wv(x) + C(v)\big), \qquad \forall\, x \in \mathbb{T}^d$$

where σ is a pointwise non-linearity and $W \in \mathbb{R}^{n\times m}$ is a learnable matrix. We further modify the layer by learning the kernel coefficients in tensorized form, adding skip connections, normalization layers, and learnable activations as outlined in Kossaifi et al., 2023. We refer the reader to this work for further details.

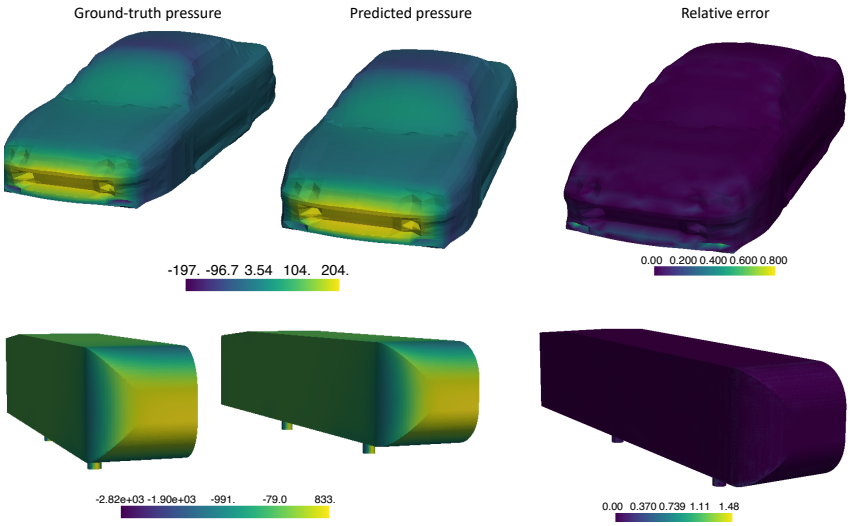


Figure 8.3: Visualization of a ground-truth pressure and corresponding prediction by GINO from the Shape-Net Car (top) and Ahmed-body (bottom) datasets, as well as the absolute error.

Adaptive instance normalization. For many engineering problems of interest, the boundary information is a fixed, scalar, inlet velocity specified on some portion of ∂D. In order to efficiently incorporate this scalar information into our architecture, we use a learnable adaptive instance normalization X. Huang and Belongie, 2017 combined with a Fourier feature embedding Sitzmann et al., 2020. In particular, the scalar velocity is embedded into a vector with Fourier features. This vector then goes through a learnable MLP, which outputs the scale and shift parameters of an instance normalization layer Ulyanov, Vedaldi, and Lempitsky, 2016. In problems where the velocity information is not fixed, we replace the normalization layers of the FNO blocks with this adaptive normalization. We find this technique improves performance, since the magnitude of the output fields usually strongly depends on the magnitude of the inlet velocity.

8.5 Experiments

We explore a range of models on two CFD datasets. The large-scale Ahmed-Body dataset, which we generated, and also the Shape-Net Car dataset from Umetani and Bickel, 2018. Both datasets contain simulations of the Reynold-Averaged Navier-Stokes (RANS) equations for a chosen turbulence model. The goal is to estimate the full pressure field given the shape of the vehicle as input. We consider GNO (Zongyi Li, Kovachki, Azizzadenesheli, B. Liu, Bhattacharya, et al., 2020b), MeshGraphNet (Pfaff et al., 2020), GeoFNO (Zongyi Li, D. Z. Huang, et al., 2022), 3D UNet (Wolny

Model	latent resolution	radius	training error	test error
GINO	32	0.055	14.11%	13.59%
GINO	48	0.055	8.99%	10.20%
GINO	64	0.055	6.00%	8.47%
GINO	80	0.055	5.77%	**7.87%**
GINO	32	0.110	8.66%	10.10%
GINO	48	0.073	7.25%	9.17%
GINO	64	0.055	6.00%	8.47%
GINO	80	0.044	6.22%	**7.89%**

When fixing the radius, larger latent resolutions lead to better performance. The gaps become smaller when fixing the number of edges and scaling the radius correspondingly.

Table 8.2: Ablation on the Ahmed-body with different sizes of the latent space

et al., 2020) with linear interpolation, FNO (Zongyi Li, Kovachki, Azizzadenesheli, B. Liu, Bhattacharya, et al., 2020a), and GINO. We train each model for 100 epochs with Adam optimizer and step learning rate scheduler. The implementation details can be found in the Appendix. All models run on a single Nvidia V100 GPU.

Ahmed-Body dataset

We generate the industry-level vehicle aerodynamics simulation based on the Ahmed-body shapes (Ahmed, Ramm, and Faltin, 1984). The shapes are parameterized with six design parameters: length, width, height, ground clearance, slant angle, and fillet radius. We also vary the inlet velocity from 10m/s to 70m/s, leading to Reynolds numbers ranging from 4.35×10^5 to 6.82×10^6. We use the GPU-accelerated OpenFOAM solver for steady state simulation using the SST $k - \omega$ turbulence model (Menter, 1993) with 7.2 million mesh points in total with 100k mesh points on the surface. Each simulation takes 7-19 hours on 2 Nvidia v100 GPUs with 16 CPU cores. We generate 551 shapes in total and divide them into 500 for training and 51 for validation.

Shape-Net Car dataset

We also consider the Car dataset generated by Umetani and Bickel, 2018. The input shapes are from the ShapeNet Car category (Chang et al., 2015). In Umetani and Bickel, 2018, the shapes are manually modified to remove the side mirrors, spoilers, and tires. The RANS equations with the $k - \epsilon$ turbulence model and SUPG stabilization are simulated to obtain the time-averaged velocity and pressure fields using a finite element solver (Zienkiewicz, Taylor, and Nithiarasu, 2013). The inlet

Model	training error	test error
GNO	18.16%	18.77%
Geo-FNO (sphere)	10.79%	15.85%
UNet (interp)	12.48%	12.83%
FNO (interp)	9.65%	9.42%
GINO (encoder-decoder)	7.95%	9.47%
GINO (decoder)	6.37%	**7.12%**

We do a benchmark study with several standard machine-learning methods on the Shape-Net and Ahmed body datasets. The training error is normalized L2 error; the test error is de-normalized L2.

Table 8.3: Shape-Net Car dataset (3.7k mesh points).

velocity is fixed at 20m/s (72km/h) and the estimated Reynolds number is 5×10^6. Each simulation takes approximately 50 minutes. The car surfaces are stored with 3.7k mesh points. We take the 611 water-tight shapes out of the 889 instances, and divide the 611 instances into 500 for training and 111 for validation.

As shown in Table 8.3 8.4 and Figure 8.3, GINO achieves the best error rate with a large margin compared with previous methods. On the Ahmed-body dataset, GINO achieves **8.31%** while the previous best method achieve 11.16%. On the Shape-Net Car, GINO achieves 7.12% error rate compared to 9.42% on FNO. It takes 0.1 seconds to evaluate, which is 100,000x faster than the GPU-parallel OpenFOAM solver that take 10 hours to generates the data. We further performance a full cost-accuracy analysis in the following section.

For ablations, we consider channel dimensions [32, 48, 64, 80], latent space [32, 48, 64, 80], and radius from 0.025 to 0.055 (with the domain size normalized to [-1, 1]). As depicted in Figure 8.5a and Table 8.2, larger latent spaces and radii yield superior performance.

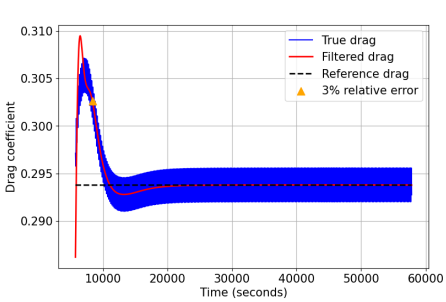
Cost-accuracy analysis on drag coefficient

To compare the performance of our model against the industry-standard OpenFOAM solver, we perform a full cost-accuracy trade-off analysis on computing the drag coefficient, which is the standard design objective of vehicles and airfoils. The result shows GINO is 26,000x faster at computing the drag coefficients. Figure 8.4b below shows the cost-accuracy curve, measured in terms of inference time needed for a relative error in the drag coefficient for GINO and OpenFOAM. The detailed setup is discussed in the appendix.

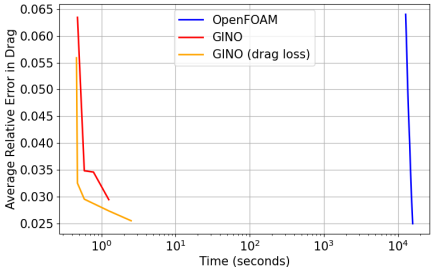
Model	training error	test error
MeshGraphNet	9.08%	13.88%
UNet (interp)	9.93%	11.16%
FNO (interp)	12.97%	12.59%
GINO (encoder-decoder)	9.36%	9.01%%
GINO (decoder)	9.34%	**8.31%**

Previous works such as GNO and Geo-FNO cannot scale to large meshes with 100k points. We instead add the MeshGraphNet for graph comparison. Again, for UNet, FNO, and GINO, we fix the latent grid to $64 \times 64 \times 64$. The training error is normalized L2; the test error is de-normalized L2.

Table 8.4: Ahmed-body dataset (100k mesh points).



(a) Example drag coefficient computed after every iteration of the OpenFOAM solver. The reference drag corresponds to the last point of the filtered signal. Triangle indicates the time when the solver reaches a 3% relative error with respect to the reference drag.



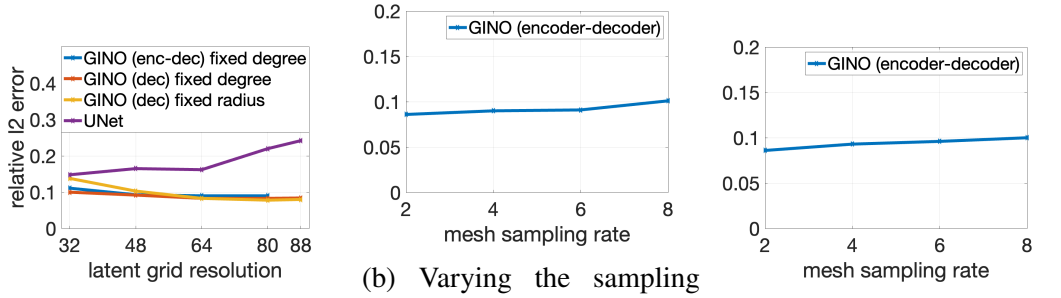
(b) Cost-accuracy trade-off curve for OpenFOAM vs. GINO. Increasing cost of GINO is achieved by increasing the size of the latent space. Loss function for models on the red line includes only error in pressure and wall shear stress, while, for models on orange line, the loss also includes drag.

Figure 8.4: Cost-accuracy trade-off analysis for the drag coefficient.

Discretization-convergence and ablation studies

We investigate discretization-convergence by varying different parts of GINO. Specifically, we vary the latent grid resolution and the sampling rates for input-output meshes. In these experiments, we fixed the training and test samples to be the same, i.e., same latent grid resolution or sampling rate, but varied the shape and input conditions.

Discretization-convergence wrt the latent grid. Here, each model is trained and tested on (the same) latent resolutions, specifically 32, 48, 64, 80, and 88, and the architecture is the same. As depicted in Figure 8.5a, GINO demonstrates a comparable error rate across all resolutions. A minor improvement in errors is



(a) Varying resolutions of the latent grid (same resolution for training and testing).

(b) Varying the sampling rates of the input-output mesh (same rate training and testing).

(c) Train with a low sampling rate and test on full mesh (zero-shot super-resolution).

Figure 8.5: Discretization-convergence studies and zero-shot super-resolution.

observed when employing a larger latent space. Conversely, the errors associated with the UNet model grow as the resolution is decreased due to the decreasing receptive field of its local convolution kernels.

Discretization-convergence in the input-output mesh. Here, GINO is trained and tested with sub-sampled input-output meshes at various sampling rates (2x, 4x, 6x, 8x). As illustrated in Figure 8.5b, GINO exhibits a consistent error rate across all sampling rates. A slight increase in errors is observed on coarser meshes.

Zero-shot super-resolution. GINO possesses the ability to perform zero-shot super-resolution. The model is trained on a coarse dataset, sub-sampled by 2x, 4x, 6x, and 8x, and subsequently tested on the full mesh, that is not seen during training. The error remains consistent across all sampling rates 8.5c. This characteristic enables the model to be trained at a coarse resolution when the mesh is dense, consequently reducing the computational requirements.

Drag Coefficient Comparison

For many engineering tasks, the goal is often to determine a single quantity of interest from a simulation which can then be used within an overall design process. In the design of automobiles, a sought after quantity is the drag coefficient of the vehicle. Intuitively, it is a number inversely proportional to the efficiency with which a vehicle passes through a fluid. Engineers are therefore often interested in designing geometries with minimal drag coefficients. For a fluid with unit density, the drag coefficient is defined as

$$c_d = \frac{2}{v^2 A}\left(\int_{\partial\Omega} p(x)\big(\hat{n}(x)\cdot\hat{i}(x)\big)\,dx + \int_{\partial\Omega} T_w(x)\cdot\hat{i}(x)\,dx\right) \tag{8.8}$$

where $\partial\Omega \subset \mathbb{R}^3$ is the surface of the car, $p : \mathbb{R}^3 \to \mathbb{R}$ is the pressure, $\hat{n} : \partial\Omega \to \mathbb{S}^2$ is the outward unit normal vector of the car surface, $\hat{i} : \mathbb{R}^3 \to \mathbb{S}^2$ is the unit direction of the inlet flow, $T_w : \partial\Omega \to \mathbb{R}^3$ is the wall shear stress on the surface of the car, $v \in \mathbb{R}$ is the speed of the inlet flow, and $A \in \mathbb{R}$ is the area of the smallest rectangle enclosing the front of the car.

For our Ahmed-body dataset, we train several GINO models (decoder) to predict the pressure on the surface of the car as well as the wall shear stress. Since the inlet flow is always parallel to the x-axis i.e. $\hat{i}(x) = (-1, 0, 0)$, we predict only the first component of the wall shear stress. Since our results from Table 8.2 indicate that varying the size of the latent space has a significant effect on predictive performance, we make this the only hyper-parameter of the model and fix all others. We choose latent resolutions varying from 24 to 86. Furthermore, we consider two different loss functions. The first is simply the average of the relative L^2 errors for pressure and wall shear stress. The second includes in this average the relative error in the drag coefficient computed using equation (8.8). The drag is always computed from our full-field predictions and is never given as part of the output from the model.

To compare the performance of our model against the industry-standard OpenFOAM solver, we perform a full cost-accuracy trade-off analysis. During data generation, we keep track of the drag coefficient predicted by OpenFOAM after every iteration. While the coefficient converges with more iterations, this convergence is not monotone and can often appear quite noisy. This makes computing the error from the raw data not possible. We therefore apply a box filter to the raw signal to compute a filtered version of the drag which acts as smoother. We take as the reference drag, the drag at the last iteration of the filtered signal. To compute the number of iterations it takes for the solver to predict a drag coefficient at a given relative error, we trace back the predictions from the filtered signal and return the first time at which this prediction incurs the given error with respect to the reference drag. An example of this methodology is shown in Figure 8.4a. The errors for our GINO model are computed with respect to the true drag coefficient from the last iteration of the solver. This is because we take as ground truth the pressure and wall shear stress from this last iteration and train our model to predict them.

Figure 8.4b shows the cost-accuracy curve, measured in terms of inference time needed for a relative error in the drag coefficient for GINO and OpenFOAM. The cost of GINO is computed as the time, averaged over the test set, needed to predict the drag coefficient by running the model. This time includes both data pre-processing

(computing the SDF) as well as the model run-time and the drag calculation given the predicted fields. All models are ran on a single NVIDIA V100 GPU. The cost for OpenFOAM is computed as described in the previous paragraph and is averaged over the test set. The solver is ran on two NVIDIA V100 GPUs in parallel. We observe a four to five order of magnitude speed-up when using GINO. At a 3% relative error, we find the speed-up from our model which includes drag in the loss to be $26,000\times$. As we increase the size of the latent space, the cost of GINO grows, however, we observe a plateau in the drag error. This is common in machine learning models as the error from using finite data starts to dominate the approximation error. Furthermore, we use only the size of the latent space as a hyper-parameter, keeping the number of learnable parameters fixed. It is interesting to explore further how parametrically scaling the model impacts predictive power.

8.6 Discussion and Conclusion

In this work, we propose the GINO model for 3D PDEs with complex geometries. The GINO model consists of the graph-kernel blocks for the encoder and decoder that go to a latent uniform space, where the Fourier blocks run on the latent space to capture the global interaction. We experiment on two CFD datasets: Shape-Net car geometries and large-scale Ahmed's body geometries, the latter encompassing over 600 car geometries featuring hundreds of thousands of mesh points. The evidence from these case studies illustrates that our method offers a substantial speed improvement, with a factor of 100,000 times acceleration in comparison to the GPU-based OpenFOAM solver. Concurrently, our approach has achieved one-fourth to one-half the error rates compared to prevailing neural networks such as 3D U-Net. This underscores the potential of our method to significantly enhance computational efficiency while maintaining a competitive level of accuracy within the realm of CFD applications. **Limitation:** The trained surrogate model is limited to a specific category of shapes. The quality of the model depends on the quality of the training dataset. For CFD with more complex shapes, it is not easy to obtain a large training dataset. We will explore physics-informed approaches (Zongyi Li, Zheng, et al., 2021) and generate time-dependent high-fidelity simulations in the future.

References

Ahmed, Syed R, G Ramm, and Gunter Faltin (1984). "Some salient features of the time-averaged ground vehicle wake". In: *SAE transactions*, pp. 473–503.

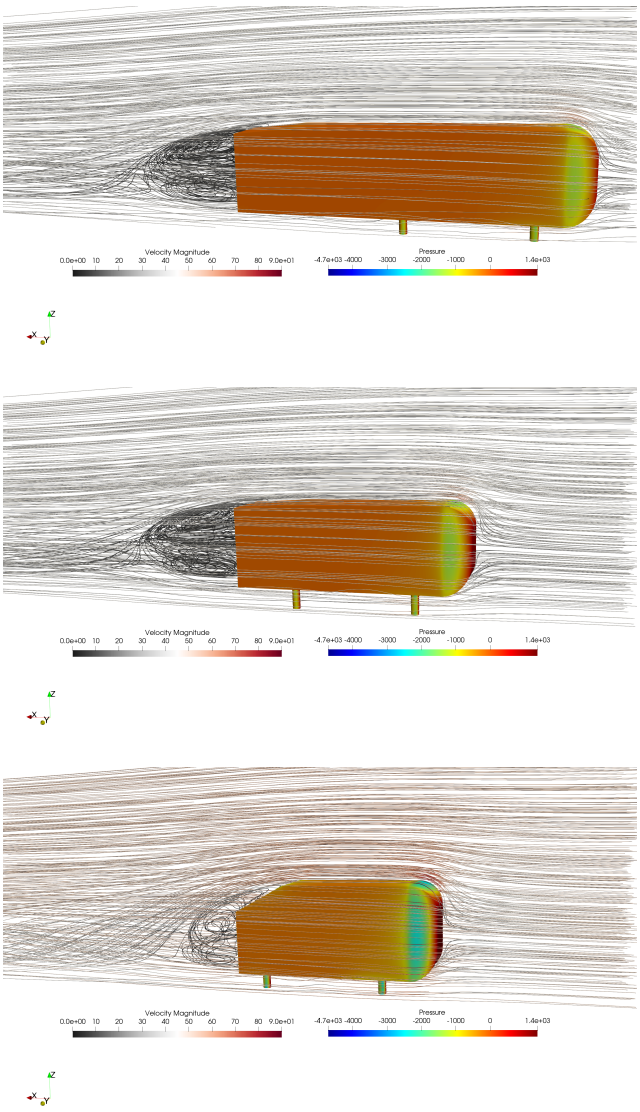


Figure 8.6: Illustrations of the Ahmed-body dataset

Alet, Ferran et al. (2019). "Graph element networks: adaptive, structured computation and memory". In: *International Conference on Machine Learning*. PMLR, pp. 212–222.

Allen, Kelsey R et al. (2022). "Physical design using differentiable learned simulators". In: *arXiv preprint arXiv:2202.00728*.

Balu, Aditya et al. (2021). "Distributed multigrid neural solvers on megavoxel domains". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14.

Battaglia, Peter et al. (2016). "Interaction networks for learning about objects, relations and physics". In: *Advances in neural information processing systems* 29.

Battaglia, Peter W et al. (2018). "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261*.

Bhattacharya, Kaushik et al. (2021). "Model reduction and neural networks for parametric PDEs". In: *The SMAI journal of computational mathematics* 7, pp. 121–157.

Brandstetter, Johannes et al. (2022). "Clifford neural layers for PDE modeling". In: *arXiv preprint arXiv:2209.04934*.

Chang, Angel X et al. (2015). "Shapenet: An information-rich 3d model repository". In: *arXiv preprint arXiv:1512.03012*.

Cohen, Albert, Christoph Schwab, and Jakob Zech (2018). "Shape Holomorphy of the Stationary Navier–Stokes Equations". In: *SIAM Journal on Mathematical Analysis* 50.2, pp. 1720–1752.

Cooley, James W and John W Tukey (1965). "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90, pp. 297–301.

Durasov, Nikita et al. (2021). "DEBOSH: Deep Bayesian Shape Optimization". In: *arXiv preprint arXiv:2109.13337*.

Fey, Matthias et al. (2023). *PyTorch Extension Library of Optimized Scatter Operations*. URL: `https://github.com/rusty1s/pytorch_scatter`.

Hamilton, Will, Zhitao Ying, and Jure Leskovec (2017). "Inductive representation learning on large graphs". In: *Advances in neural information processing systems* 30.

Hao, Zhongkai et al. (2023). "GNOT: A General Neural Operator Transformer for Operator Learning". In: *arXiv preprint arXiv:2302.14376*.

Hermosilla, Pedro et al. (2018). "Monte carlo convolution for learning on non-uniformly sampled point clouds". In: *ACM Transactions on Graphics (TOG)* 37.6, pp. 1–12.

Huang, Xun and Serge J. Belongie (2017). "Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization". In: *CoRR* abs/1703.06868.

Jasak, Hrvoje, Aleksandar Jemcov, Zeljko Tukovic, et al. (2007). "OpenFOAM: A C++ library for complex physics simulations". In: *International workshop on coupled methods in numerical dynamics*. Vol. 1000, pp. 1–20.

Kipf, Thomas N and Max Welling (2016). "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907*.

Kissas, Georgios et al. (2022). "Learning operators with coupled attention". In: *Journal of Machine Learning Research* 23.215, pp. 1–63.

Korzun, Ashley M et al. (2022). "Application of a Detached Eddy Simulation Approach with Finite-Rate Chemistry to Mars-Relevant Retropropulsion Operating Environments". In: *AIAA SciTech 2022 Forum*, p. 2298.

Kossaifi, Jean et al. (2023). *Multi-Grid Tensorized Fourier Neural Operator for High-Resolution PDEs*. arXiv: 2310.00120.

Kovachki, Nikola et al. (2021). "Neural operator: Learning maps between function spaces". In: *arXiv preprint arXiv:2108.08481*.

Lam, Remi et al. (2022). "GraphCast: Learning skillful medium-range global weather forecasting". In: *arXiv preprint arXiv:2212.12794*.

Lee, Seungjun (n.d.). "Mesh-Independent Operator Learning for Partial Differential Equations". In: *ICML 2022 2nd AI for Science Workshop*.

Li, Yangyan et al. (2018). "Pointcnn: Convolution on x-transformed points". In: *Advances in neural information processing systems* 31.

Li, Zijie, Kazem Meidani, and Amir Barati Farimani (2022). "Transformer for partial differential equations' operator learning". In: *arXiv preprint arXiv:2205.13671*.

Li, Zongyi, Daniel Zhengyu Huang, et al. (2022). "Fourier neural operator with learned deformations for pdes on general geometries". In: *arXiv preprint arXiv:2207.05209*.

Li, Zongyi, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, et al. (2020a). "Fourier Neural Operator for Parametric Partial Differential Equations". In: *arXiv preprint arXiv:2010.08895*.

– (2020b). "Neural operator: Graph kernel network for partial differential equations". In: *arXiv preprint arXiv:2003.03485*.

Li, Zongyi, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, et al. (2020). "Multipole graph neural operator for parametric partial differential equations". In: *Advances in Neural Information Processing Systems* 33.

Li, Zongyi, Hongkai Zheng, et al. (2021). "Physics-informed neural operator for learning partial differential equations". In: *arXiv preprint arXiv:2111.03794*.

Lin, Haitao et al. (2022). "Non-equispaced Fourier Neural Solvers for PDEs". In: *arXiv preprint arXiv:2212.04689*.

Liu, Ning, Siavash Jafarzadeh, and Yue Yu (2023). "Domain Agnostic Fourier Neural Operators". In: *arXiv preprint arXiv:2305.00478*.

Menter, Florianr (1993). "Zonal two equation kw turbulence models for aerodynamic flows". In: *23rd fluid dynamics, plasmadynamics, and lasers conference*, p. 2906.

Pathak, Jaideep et al. (2022). "Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators". In: *arXiv preprint arXiv:2202.11214*.

Pfaff, Tobias et al. (2020). "Learning mesh-based simulation with graph networks". In: *arXiv preprint arXiv:2010.03409*.

Rahman, Md Ashiqur, Zachary E Ross, and Kamyar Azizzadenesheli (2022). "U-no: U-shaped neural operators". In: *arXiv preprint arXiv:2204.11127*.

Remelli, Edoardo et al. (2020). "Meshsdf: Differentiable iso-surface extraction". In: *Advances in Neural Information Processing Systems* 33, pp. 22468–22478.

Sanchez-Gonzalez, Alvaro et al. (2020). "Learning to simulate complex physics with graph networks". In: *International conference on machine learning*. PMLR, pp. 8459–8468.

Sitzmann, Vincent et al. (2020). "Implicit neural representations with periodic activation functions". In: *Advances in Neural Information Processing Systems* 33, pp. 7462–7473.

Stein, Elias M. (1970). *Singular Integrals and Differentiability Properties of Functions*. Princeton University Press.

Sun, Hongyu et al. (2022). "Accelerating Time-Reversal Imaging with Neural Operators for Real-time Earthquake Locations". In: *arXiv preprint arXiv:2210.06636*.

Temam, R. (1984). *Navier-Stokes Equations: Theory and Numerical Analysis*. Studies in mathematics and its applications. North-Holland.

Ulyanov, Dmitry, Andrea Vedaldi, and Victor Lempitsky (2016). "Instance normalization: The missing ingredient for fast stylization". In: *arXiv preprint arXiv:1607.08022*.

Umetani, Nobuyuki and Bernd Bickel (2018). "Learning three-dimensional flow for interactive aerodynamic design". In: *ACM Transactions on Graphics (TOG)* 37.4, pp. 1–10.

Wandel, Nils, Michael Weinmann, and Reinhard Klein (2021). "Teaching the incompressible Navier–Stokes equations to fast neural surrogate models in three dimensions". In: *Physics of Fluids* 33.4.

Wen, Gege et al. (2023). "Real-time high-resolution CO 2 geological storage prediction using nested Fourier neural operators". In: *Energy & Environmental Science* 16.4, pp. 1732–1741.

Whitney, Hassler (1934). "Analytic Extensions of Differentiable Functions Defined in Closed Sets". In: *Transactions of the American Mathematical Society* 36.1, pp. 63–89.

Wolny, Adrian et al. (July 2020). "Accurate and versatile 3D segmentation of plant tissues at cellular resolution". In: *eLife* 9. Ed. by Christian S Hardtke et al., e57613. ISSN: 2050-084X. DOI: 10.7554/eLife.57613. URL: https://doi.org/10.7554/eLife.57613.

Wu, Wenxuan, Zhongang Qi, and Li Fuxin (2019). "Pointconv: Deep convolutional networks on 3d point clouds". In: *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 9621–9630.

Yang, Yan et al. (2021). "Seismic wave propagation and inversion with neural operators". In: *The Seismic Record* 1.3, pp. 126–134.

Zhou, Qian-Yi, Jaesik Park, and Vladlen Koltun (2018). "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847*.

Zienkiewicz, Olek C, Robert Leroy Taylor, and Perumal Nithiarasu (2013). *The finite element method for fluid dynamics*. Butterworth-Heinemann.

Chapter 9

GEOMETRY: NEURAL OPERATOR WITH LEARNED DEFORMATION

Deep learning surrogate models have shown promise in solving partial differential equations (PDEs). Among them, the Fourier neural operator (FNO) achieves good accuracy, and is significantly faster compared to numerical solvers, on a variety of PDEs, such as fluid flows. However, the FNO uses the Fast Fourier transform (FFT), which is limited to rectangular domains with uniform grids. In this work, we propose a new framework, viz., Geo-FNO, to solve PDEs on arbitrary geometries. Geo-FNO learns to deform the input (physical) domain, which may be irregular, into a latent space with a uniform grid. The FNO model with the FFT is applied in the latent space. The resulting Geo-FNO model has both the computation efficiency of FFT and the flexibility of handling arbitrary geometries. Our Geo-FNO is also flexible in terms of its input formats, viz., point clouds, meshes, and design parameters are all valid inputs. We consider a variety of PDEs such as the Elasticity, Plasticity, Euler's, and Navier-Stokes equations, and both forward modeling and inverse design problems. Comprehensive cost-accuracy experiments show that Geo-FNO is 10^5 times faster than the standard numerical solvers and twice more accurate compared to direct interpolation on existing ML-based PDE solvers such as the standard FNO.

9.1 Introduction

Data-driven engineering design has the potential to accelerate the design process by orders of magnitude compared to conventional methods. It can enable extensive exploration of the design space and yield new designs with far greater efficiency. This is because the conventional design process requires repeated evaluations of partial differential equations (PDEs) for optimization, which can be time-consuming. Examples include computational fluid dynamics (CFD) based aerodynamic design and topology optimization for additive manufacturing. Deep learning approaches have shown promise in speeding up PDE evaluations and automatically computing derivatives, hence accelerating the overall design cycle. However, most deep learning approaches currently focus on predicting a few important quantities in the design process, e.g. lift and drag in aerodynamics, as opposed to completely emulating the entire simulation (i.e., pressure or Mach number fields in aerodynamics). This limits

the design space to be similar to the training data, and may not generalize to new geometries and scenarios. In contrast, we develop an efficient deep learning-based approach for design optimization, which emulates the entire PDE simulation on general geometries leading to a versatile tool for exploring the design space.

Deformable meshes. Solutions of PDEs frequently have high variations across the problem domain, and hence some regions of the domain often require a finer mesh compared to other regions for obtaining accurate solutions. For example, in airfoil flows, the region near the airfoil requires a much higher resolution for accurately modeling the aerodynamics, as shown in Figure 9.8. To address such variations, deformable mesh methods such as adaptive finite element methods (FEM) have been developed. Two adaptive mesh methods are commonly used: the adaptive mesh refinement method that adds new nodes (and higher-order polynomials), and the adaptive moving mesh method that relocates the existing nodes Babuška and Suri, 1990; W. Huang and Russell, 2010. While mesh refinement is popular and easy to use with traditional numerical solvers, it changes the mesh topology and requires special dynamic data structures, which reduce speed and make it hard for parallel processing. On the other hand, the adaptive moving mesh methods retain the topology of the mesh, making it possible to integrate with spectral methods. Spectral methods solve the equation on the spectral space (i.e., Fourier, Chebyshev, and Bessel), which usually have exponential convergence guarantees when applicable Gottlieb and Orszag, 1977. However, these spectral methods are limited to simple computational domains with uniform grids. When the mesh becomes non-uniform, spectral basis functions are no longer orthogonal and the spectral transform is no longer invertible, so the system in the deformed Fourier space is not equivalent to the original one anymore. Thus, traditional numerical methods are slow on complex geometries due to computational constraints.

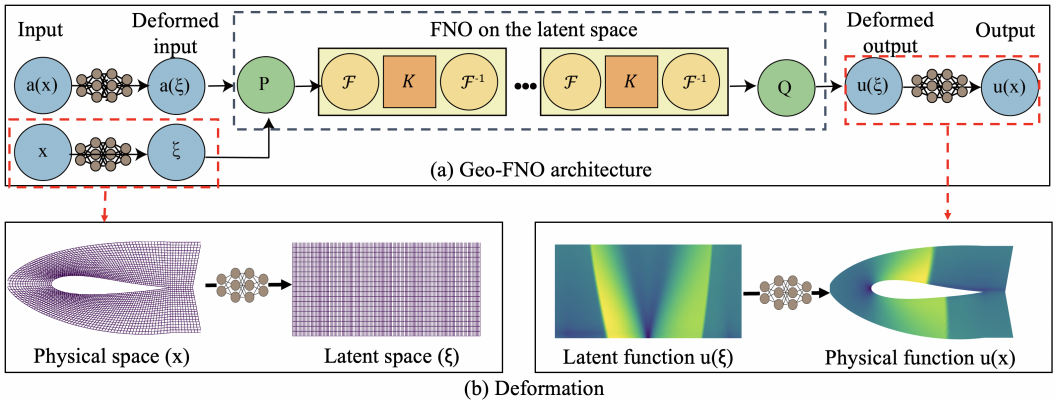
Neural operators. Deep learning surrogate models have recently yielded promising results in solving PDEs. One class of such models is data-driven, and they directly approximate the input-output map through supervised learning. Such models have achieved significant speedups compared to traditional solvers in numerous applications (Zhu and Zabaras, 2018; Bhatnagar et al., 2019). Among them, the graph neural networks have not been studied for complex geometries (Allen et al., 2022; Sanchez-Gonzalez et al., 2020) Recently, a new class of data-driven models known as neural operators aim to directly learn the solution operator of the

PDE in a mesh-convergent manner. Unlike standard deep learning models, such as convolutional neural networks from computer vision, neural operators are consistent to discretization and hence, better suited for solving PDEs. They generalize the previous finite-dimensional neural networks to learn operator mapping between infinite-dimensional function spaces. Examples include Fourier neural operator (FNO) (Li, Kovachki, et al., 2020c), graph neural operator Li, Kovachki, et al., 2020b, and DeepONet (Lu, Jin, and Karniadakis, 2019). We consider FNO (Li, Kovachki, et al., 2020a) in this paper due to its superior cost-accuracy trade-off (De Hoop et al., 2022). FNO implements a series of layers computing global convolution operators with the fast Fourier transform (FFT) followed by mixing weights in the frequency domain and inverse Fourier transform. These global convolutional operators are interspersed with non-linear transformations such as GeLU. By composing global convolution operators and non-linear activations, FNO can approximate highly non-linear and non-local solution operators. FNO and its variants are able to simulate many PDEs such as the Navier-Stokes equation and seismic waves, do high-resolution weather forecasts, and predict CO2 migration with unprecedented cost-accuracy trade-off (Pathak et al., 2022; Yang et al., 2021; Wen et al., 2022).

Limitations of FNO on irregular geometries. While FNO is fast and accurate, it has limitations on the input format and the problem domain. Since FNO is implemented with FFT, it can be only applied on rectangular domains with uniform meshes. When applying it to irregular domain shapes, previous works usually embed the domain into a larger rectangular domain (Lu, Meng, et al., 2022). However, such embeddings are less efficient and wasteful, especially for highly irregular geometries. Similarly, if the input data is in the form of non-uniform meshes such as triangular meshes, previous works use basic interpolation methods between the input non-uniform mesh and a uniform mesh on which FFT is computed Li, Kovachki, et al., 2020a. This can cause large interpolation errors, especially for non-linear PDEs.

Our contributions. To address the above challenges, we develop a geometry-aware Fourier neural operator (Geo-FNO) and we summarize our contributions below:

1. We propose a geometry-aware discretization-convergent FNO framework



Top: The Geo-FNO model deforms the input functions from irregular physical space to a uniform latent space. After the standard FNO Li, Kovachki, et al., 2020a is applied, the Geo-FNO will deform the latent solution function back to the physical space. P, Q are lifting and projection. $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier transforms. K is a linear transform (9.18). **Bottom:** The deformation, either given or learned, defines a correspondence between the physical space and computational space. It induces an adaptive mesh and a generalized Fourier basis on the physical space.

Figure 9.1: Geometry-aware FNO

(Geo-FNO) that works on arbitrary geometries and a variety of input formats such as point clouds, non-uniform meshes, and design parameters.

2. Geo-FNO deforms the irregular input domain into a uniform latent mesh on which the FFT can be applied. Such deformation can be fixed or learned with the FNO architecture in an end-to-end manner. For the latter case, we design a neural network to model the deformation and train it end-to-end with Geo-FNO.

3. We experiment on different geometries for the Elasticity, Plasticity, Advection, Euler's, and Navier-Stokes equations on both forward modeling and inverse design tasks. The cost-accuracy experiments show that Geo-FNO has up to 10^5 acceleration compared to the numerical solver, as well as half the error rate compared to previous interpolation-based methods as shown in Figure 9.3. Further, Geo-FNO maintains discretization-convergence. The model trained on low resolution datasets can be directly evaluated at high resolution cases.

Geo-FNO thus learns a deformable mesh along with the solution operator in an end-to-end manner by applying a series of FFTs and inverse FFTs on a uniform latent mesh, as shown in Figure 9.1. Thus, Geo-FNO combines both the computational

efficiency of the FFT and the flexibility of learned deformations. It is as fast as the original FNO but can represent the variations in solutions and domain shapes more efficiently and accurately. The learned deformation from the physical irregular domain to a uniform computational domain is inspired by the traditional adaptive moving mesh method (W. Huang and Russell, 2010). However, the adaptive moving mesh method has not had much success with traditional numerical solvers due to the loss of orthogonality of the spectral transform. On a deformed mesh, the system in the Fourier space is no longer equivalent to the original system, so it does not make sense to solve the PDE in the deformed Fourier space. However, Geo-FNO does not have this limitation. It does not involve solving equations in the Fourier space. Instead, Geo-FNO approximates the solution operator via the computation on Fourier space in a data-driven manner.

In principle, our Geo-FNO framework can be directly extended to general topologies. Given complex input topology we can apply the decomposition techniques such as triangle tessellation to divide the domain into sub-domains such as each sub-domain is regular. Similarly, for other input formats such as the signed distance functions (SDF), we can sample collocation points. Furthermore, it is possible to extend the Geo-FNO framework to the physics-informed neural operator setting which incorporates PDE constraints (Li, Zheng, et al., 2021). Since the deformation is parameterized by a neural network, we can obtain the exact derivatives and apply the chain rule to compute the residual error.

9.2 Problem Settings and Preliminaries

Problem settings. In this work, we consider parametric partial differential equations defined on various domains. Assume the problem domain D_a is parameterized by design parameters $a \in \mathcal{A}$, which follows some distribution $a \sim \mu$. For simplicity, we assume all the domains are bounded, orientable manifolds embedded in some background Euclidean space Ω (e.g., $\mathbb{R}^3$). We consider both stationary problems and time-dependent problems of the following forms:

$$\begin{aligned} \frac{du}{dt} &= \mathcal{R}(u), && \text{in } D_a \times T \\ u &= u_0, && \text{in } D_a \times \{0\} \\ u &= u_b, && \text{in } \partial D_a \times T \end{aligned} \tag{9.1}$$

where $u_0 \in \mathcal{U}(0)$ is the initial condition; u_b is the boundary condition; and $u(t) \in \mathcal{U}(t)$ for $t > 0$ is the target solution function. $\mathcal{R}$ is a possibly non-linear differential operator. We assume that u exists and is bounded for all time and for every $u_0 \in \mathcal{U}(t)$

at every $t \in T$. This formulation gives rise to the solution operator $\mathcal{G}^\dagger : (a, u_0, u_b) \mapsto u$. Prototypical examples include fluid mechanics problems such as the Burgers' equation and the Navier-Stokes equation and solid mechanics problems such as elasticity and plasticity defined on various domain shapes such as airfoils and pipes. In this paper, we assume the initial condition u_0 and the boundary condition u_b are fixed. Specifically, we also consider stationary problem $\mathcal{R}(u) = 0$ where u_0 is not necessary. In this case, the solution operator simplifies to $\mathcal{G}^\dagger : a \mapsto u$.

Input formats. In practice, the domain shape can be parameterized in various ways. It can be given as meshes, functions, and design parameters. In the most common cases, the domain space is given as meshes (point clouds) $a = \mathcal{T} = \{x_i\}^N \subset \Omega$, such as triangular tessellation meshes used in many traditional numerical solvers. Alternatively, the shapes can be described by some functions $a = f : \Omega \to \mathbb{R}$. For example, if the domain is a 2D surface, then it can be parameterized by its boundary function $f : [0, 1] \to \partial D_a$. Similarly, the domain can be described as signed distance functions (SDF) or occupancy functions. These are common examples used in computer vision and graphics. At last, we also consider specific design parameters $a \in \mathbb{R}^d$. For example, it can be the width, height, volume, angle, radius, and spline nodes used to model the boundary. The design parameters restrict the domain shape in a relatively smaller subspace of all possible choices. This format is most commonly used in a design problem, which is easy to optimize and manufacture. For the latter two cases, it is possible to sample a mesh $\mathcal{T}$ based on the shape.

Learning the solution operator

Given a PDE as defined in (9.1) and the corresponding solution operator $\mathcal{G}^\dagger$, one can use a neural operator $\mathcal{G}_\theta$ with parameters θ as a surrogate model to approximate $\mathcal{G}^\dagger$. Usually we assume a dataset $\{a_j, u_j\}_{j=1}^N$ is available, where $\mathcal{G}^\dagger(a_j) = u_j$ and $a_j \sim \mu$ are i.i.d. samples from some distribution μ supported on $\mathcal{A}$. In this case, one can optimize the solution operator by minimizing the relative empirical data loss on a given data pair

$$\mathcal{L}_{\text{data}}(u, \mathcal{G}_\theta(a)) = \frac{\|u - \mathcal{G}_\theta(a)\|_{\mathcal{L}^2}}{\|u\|_{\mathcal{L}^2}} = \sqrt{\frac{\int_{D_a} |u(x) - \mathcal{G}_\theta(a)(x)|^2 \mathrm{d}x}{\int_{D_a} u(x)^2 \mathrm{d}x}} \tag{9.2}$$

The operator data loss is defined as the average error across all possible inputs

$$\mathcal{J}_{\text{data}}(\mathcal{G}_\theta) = \mathbb{E}_{a\sim\mu}[\mathcal{L}_{\text{data}}(\mathcal{G}^\dagger(a), \mathcal{G}_\theta(a))] \approx \frac{1}{N}\sum_{j=1}^{N}\sqrt{\frac{\int_{D_a}|u_j(x) - \mathcal{G}_\theta(a_j)(x)|^2\mathrm{d}x}{\int_{D_a}u_j(x)^2\mathrm{d}x}}. \tag{9.3}$$

When the prediction u is time-dependent, the $\mathcal{L}^2$ integration should also account for the temporal dimension.

Neural operator

In this work, we will focus on the neural operator model designed for the operator learning problem. The neural operator, proposed in (Li, Kovachki, et al., 2020c), is formulated as a generalization of standard deep neural networks to operator setting. Neural operators compose linear integral operator $\mathcal{K}$ with pointwise non-linear activation function σ to approximate highly non-linear operators.

Definition 9.2.1 (Neural operator $\mathcal{G}_\theta$) *Define the neural operator*

$$\mathcal{G}_\theta \coloneqq \mathcal{Q} \circ (W_L + \mathcal{K}_L + b_L) \circ \cdots \circ \sigma(W_1 + \mathcal{K}_1 + b_1) \circ \mathcal{P} \tag{9.4}$$

where $\mathcal{P} : \mathbb{R}^{d_a} \to \mathbb{R}^{d_1}$, $\mathcal{Q} : \mathbb{R}^{d_L} \to \mathbb{R}^{d_u}$ are the pointwise neural networks that encode the lower dimension function into higher dimensional space and vice versa. The model stack L layers of $\sigma(W_l + \mathcal{K}_l + b_l)$ where $W_l \in \mathbb{R}^{d_{l+1}\times d_l}$ are pointwise linear operators (matrices), $\mathcal{K}_l : \{D \to \mathbb{R}^{d_l}\} \to \{D \to \mathbb{R}^{d_{l+1}}\}$ are integral kernel operators, $b_l : D \to \mathbb{R}^{d_{l+1}}$ are the bias term made of a linear layer, and σ are fixed activation functions. The parameters θ consists of all the parameters in $\mathcal{P}, \mathcal{Q}, W_l, \mathcal{K}_l, b_l$.

Definition 9.2.2 (Fourier integral operator $\mathcal{K}$) *Define the Fourier integral operator*

$$\big(\mathcal{K}(\phi)v_t\big)(x) = \mathcal{F}^{-1}\Big(R_\phi \cdot (\mathcal{F}v_t)\Big)(x) \qquad \forall x \in D \tag{9.5}$$

where R_ϕ is the Fourier transform of a periodic function $\kappa : \bar{D} \to \mathbb{R}^{d_v\times d_v}$ parameterized by $\phi \in \Theta_{\mathcal{K}}$.

The Fourier transform $\mathcal{F}$ is defined as

$$(\mathcal{F}v)(k) = \langle v, \psi(k)\rangle_{L(D)} = \int_x v(x)\psi(x,k)\mu(x) \approx \sum_{x\in\mathcal{T}} v(x)\psi(x,k) \tag{9.6}$$

where $\psi(x,k) = e^{2i\pi\langle x,k\rangle} \in L(D)$ is the Fourier basis and $\mathcal{T}$ is the mesh sampled from the distribution μ. In Li, Kovachki, et al., 2020c, the domain D is assumed to be a periodic, square torus and the mesh $\mathcal{T}$ is uniform, so the Fourier transform $\mathcal{F}$ can be implemented by the fast Fourier transform algorithm (FFT). In this work, we aim to extend the framework to non-uniform meshes and irregular domains.

9.3 Geometry-Aware Fourier Neural Operator

The idea of the geometry-aware Fourier neural operator is to deform the physical space into a regular computational space so that the fast Fourier transform can be performed on the computational space. Formally, we want to find a diffeomorphic deformation ϕ_a between the input domain D_a and the unit torus $D^c = [0,1]^d$. The computational mesh D^c is shared among all input space D_a. It is equipped with a uniform mesh and standard Fourier basis. Once the coordinate map ϕ_a is determined, the map induces an adaptive mesh and deformed Fourier basis on the physical space. In the community of numerical PDEs, the diffeomorphism corresponds to the adaptive moving mesh (W. Huang and Russell, 2010).

Deformation from the physical space to the computational space.

Let D_a be the physical domain and D^c be the computational domain. Let $x \in D_a$ and $\xi \in D^c$ be the corresponding mesh points. Denote the input meshes as $\mathcal{T}_a = \{x^{(i)}\} \subset D_a$ and the computational meshes as $\mathcal{T}^c = \{\xi^{(i)}\} \subset D^c$. For simplicity, we assume the output mesh is the same as the input mesh. The adaptive moving mesh is defined by a coordinate transformation ϕ_a that transforms the points from the computational mesh to the physical mesh, as shown in Figure 9.1 (b)

$$\begin{aligned} \phi_a : D^c &\to D_a \\ \xi &\mapsto x \end{aligned} \tag{9.7}$$

Ideally, ϕ_a is a diffeomorphism, meaning it has an inverse ϕ_a^{-1}, and both ϕ_a and ϕ_a^{-1} are smooth. When such a diffeomorphism exists, there is a correspondence between the physical space and the computational space. Let $\mathcal{T}^c \subset D^c$ be the uniform mesh and $\psi^c \in L(D^c)$ be the standard spectral basis defined on the computational space. The coordinate transformation $\phi_a \subset D_a$ induces an adaptive mesh $\mathcal{T}_a$ and adaptive spectral basis $\psi_a \in L(D_a)$ (pushforward)

$$\begin{aligned} \mathcal{T}_a &\coloneqq \phi_a(\mathcal{T}^c) \\ \psi_a(x) &\coloneqq \psi^c \circ \phi_a^{-1}(x) \end{aligned} \tag{9.8}$$

It can be interpreted as solving the system $\mathcal{R}(v) = 0$ with adaptive mesh $\mathcal{T}_a$ and generalized basis ψ_a. Conversely, for any function defined on the physical domain $v \in L(D_a)$, it can be transformed to the computational domain $v^c \in L(D^c)$ (pullback)

$$v^c(\xi) \coloneqq v(\phi_a(\xi)) \tag{9.9}$$

Similarly, for any system of equations $\mathcal{R}(v) = 0$ such as (9.1) defined on the physical domain D_a, the transformation ϕ_a induced a new deformed system of equations $\mathcal{R}^c(v^c) = 0$. It can be interpreted as solving the deformed system $\mathcal{R}^c(v^c) = 0$ with uniform mesh $\mathcal{T}^c$ and standard basis ψ^c.

Examples: Chebyshev Basis. Chebyshev method is a standard spectral method for solving PDE with non-periodic boundary Driscoll, Hale, and Trefethen, 2014. It can be induced from the standard cosine basis (in discrete cosine transform) with a cosine grid. The Chebyshev polynomials on domain $[-1, 1]$ has the form of

$$\psi^k_{\text{cheb}} := \cos(k \cdot \cos^{-1}(x)), \quad k = 0, 1, 2 \dots$$

It's equivalent to apply the cosine deformation $\phi_a : [0, \pi] \to [-1, 1]$

$$\phi_a(x) = \cos(x)$$

to the basis of cosine series

$$\psi^k_{\cos} := \cos(k \cdot x), \quad k = 0, 1, 2 \dots$$

in the same manner as a deformed basis (9.8)

$$\psi^k_{\text{cheb}} = \psi^k_{\cos} \circ \phi_a^{-1}(x)$$

Intuitively, the cosine deformation places more grid points around the boundary -1 and 1, which addresses the stability issue around the boundary. Notice, in practice the discrete cosine transform on $[0, \pi]$ can be implemented by the faster Fourier transform using an extended domain $[-\pi, \pi]$ and restricts to real coefficients. We will discuss the use of extended domain in Section 9.3 Fourier continuation. Therefore, GeoFNO equipped with the cosine deformation can implement the Chebyshev method. However, it is hard to deform the Fourier basis into spherical basis, since the underlying domain is not homeomorphic. For these non-homeomorphic domain we will use domain decomposition as discussed in Section 9.3.

Geometric Fourier transform

Based on the deformation, we can define the spectral transform in the computational space. Fourier transforms conducted in the computational domain are standard, since we have a uniform structured mesh in the computational domain. In this subsection, We will introduce a geometric spectral transform between the function $v(x)$ in the physical domain D_a and the function $\hat{v}(k)$ in the spectral space of the computation domain D^c.

To transform the function $v(x)$ from the physical domain to the spectral space of the computation domain, we define the forward transform:

$$(\mathcal{F}_a v)(k) := \int_{D^c} v^c(\xi) e^{-2i\pi\langle \xi, k\rangle} d\xi \tag{9.10}$$

$$= \int_D v(x) e^{-2i\pi\langle \phi_a^{-1}(x), k\rangle} |\det[\nabla_x \phi_a^{-1}(x)]| dx \tag{9.11}$$

$$\approx \frac{1}{|\mathcal{T}_a|} \sum_{x\in\mathcal{T}_a} m(x) v(x) e^{-2i\pi\langle \phi_a^{-1}(x), k\rangle} \tag{9.12}$$

where the weight $m(x) = |\det[\nabla_x \phi_a^{-1}(x)]|/\rho_a(x)$ and ρ_a is a distribution from which the input mesh $\mathcal{T}_a$ is sampled. Notice, in many cases, we have an input mesh $\mathcal{T}_a$ so that computational mesh $\phi_a^{-1}(\mathcal{T}_a) = T^c$ is uniform. In this case, $|\det[\nabla_x \phi_a^{-1}(x)]| = \rho_a(x)$, and $m(x) = 1$ can be omitted. Other choices can be made, for example, we can define $m(x)$ using heuristics depending on the solution $u(x)$ or residual error. The weight $m(x)$ corresponds to the monitor functions in the literature of adaptive meshes, where the adaptive mesh is finer near x, when $m(x)$ is large.

To transform the spectral function $\hat{v}(k)$ from the spectral space of the computation domain to the physical domain, we define the inverse transform

$$(\mathcal{F}_a^{-1}\hat{v})(x) = (\mathcal{F}^{-1}\hat{v})(\phi_a^{-1}(x)) = \sum_k \hat{v}(k) e^{2i\pi\langle \phi_a^{-1}(x), k\rangle} \tag{9.13}$$

It is worth mentioning that $\mathcal{F}_a \circ \mathcal{F}_a^{-1} = I$, since both are defined on the computational space.

$$(\mathcal{F}_a \circ \mathcal{F}_a^{-1}\hat{v})(k) = \int_{D^c} (\mathcal{F}_a^{-1}\hat{v})^c(\xi) e^{-2i\pi\langle \xi, k\rangle} d\xi \tag{9.14}$$

$$= \int_{D^c} \sum_k \hat{v}(k) e^{2i\pi\langle \xi, k\rangle} e^{-2i\pi\langle \xi, k\rangle} d\xi \quad = \hat{v}(k) \tag{9.15}$$

Notice both $\mathcal{F}_a$ (9.12) and $\mathcal{F}_a^{-1}$ (9.13) only involve the inverse coordinate transform $\phi_a^{-1} : D_a \to D^c$. Intuitively, in the forward Fourier transform $\mathcal{F}_a$, we use ϕ_a^{-1} to

transform the input function to the computational space, while in the inverse Fourier transform $\mathcal{F}_a^{-1}$, we use ϕ_a^{-1} to transform the query point to the computational space to evaluate the Fourier basis. It makes the implementation easy in that we only need to define ϕ_a^{-1}.

Architecture of Geo-FNO and the deformation neural network

We consider two scenarios: (1) the coordinate map is given, and (2) learning a coordinate map. In many cases, the input mesh $\mathcal{T}_a$ is non-uniform but structured. For example, the meshes of airfoils are usually generated in the cylindrical structure. If the input mesh is structured, meaning it can be indexed as a multi-dimensional array $\mathcal{T}_a[i_1, \ldots, i_d]$ with $0 \le i_k \le s_k \ \forall k$, then its indexing induces a canonical coordinate map

$$\phi_a^{-1} : \mathcal{T}_a[i_1, \ldots, i_d] \mapsto (i_1/s_1, \ldots, i_d/s_d) \tag{9.16}$$

In this case, $(i_1/s_1, \ldots, i_d/s_d)$ forms a uniform mesh within a unit cube, allowing for the direct application of the FFT. And Geo-FNO reduces to a standard FNO directly applied to the input meshes.

When we need to learn a coordinate map, we parameterize the coordinate map ϕ_a^{-1} as a neural network and learn it along with the solution operator G_θ in an end-to-end manner with loss (9.3).

$$\phi_a^{-1} : (x_1, x_2, \cdots, x_d, a) \mapsto (\xi_1, \xi_2, \cdots, \xi_d) \tag{9.17}$$

We parameterize the deformation as $\phi_a^{-1}(x, a) = x + f(x, a)$, where f is a standard feed-forward neural network. Specially, f takes input $(x_1, x_2, \cdots, x_d, a)$, where a is the geometry parameters. This formulation initializes ϕ_a^{-1} around an identity map, which is a good initial choice. Following the works of implicit representation (Mildenhall et al., 2020), (Sitzmann et al., 2020), we use sinusoidal features in the first layers $\sin(Bx)$, $B = 2^i$ to improve the expressiveness of the network. Concretely, we define f as a three-layer feed-forward neural network with a width of 32. The input to this network comprises a combination of components, namely x and a, alongside a series of trigonometric terms such as $\sin(2^1 x)$, $\cos(2^1 x)$, $\sin(2^2 x)$, $\cos(2^2 x)$, $\cdots$, $\sin(2^K x)$ and $\cos(2^K x)$, which are applied coordinatewisely. And a can represent the coordinates of the point clouds that encode the design geometry. Finally, ϕ_a^{-1} is used to compute $\mathcal{F}_a$ and $\mathcal{F}_a^{-1}$ (See (9.12) and (9.13)).

As shown in Figure 9.1, Geo-FNO has a similar architecture as the standard FNO,

but with a deformation in the input and output

$$\mathcal{G}_\theta \coloneqq \mathcal{Q} \circ (W_L + \mathcal{K}_L(\phi_a) + b_L) \circ \cdots \circ \sigma(W_1 + \mathcal{K}_1(\phi_a) + b_1) \circ \mathcal{P} \tag{9.18}$$

Here, we begin by employing $\mathcal{P}$ to lift (encode) the input to a higher channel dimension. Subsequently, the first Fourier convolution operator $\mathcal{K}_1$ employs the geometric Fourier transform $\mathcal{F}_a$ (9.12), while the last layer $\mathcal{K}_L$ employs the inverse geometric Fourier transform $\mathcal{F}_a^{-1}$ (9.13). To finalize the process, we utilize $\mathcal{Q}$ to project (decode) the output back to the desired dimension. In the case where the input is presented as a structured mesh, we define the coordinates as (9.16). Under this circumstance, both $\mathcal{F}_a$ and $\mathcal{F}_a^{-1}$ reduce to the standard FFT, leading to Geo-FNO being reduced to the standard FNO.

Fourier continuation

For some PDE problems, the input domain has an irregular topology that is not homeomorphic to a disk or a torus. Consequentially, there does not exist a diffeomorphism between D_a and D^c. In this case, we will first embed the input domain into a larger regular domain

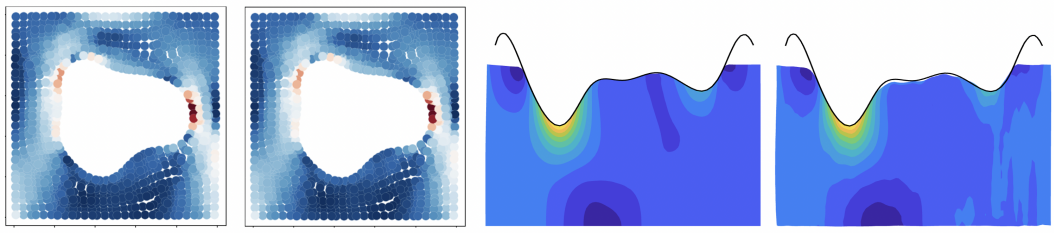
$$D_a \overset{i}{\hookrightarrow} \bar{D}_a$$

so that $\bar{D}_a$ is diffeomorphic to D^c. For example, the elasticity problem section 9.4 has a hole in its square domain. We can first embed the domain into a full square by completing the hole. Such embedding corresponds to the Fourier continuation (Bruno, Han, and Pohlman, 2007) techniques used in conventional spectral solvers. Usually it requires to extend the function $u \in \mathcal{U}(D_a)$ to $\bar{u} \in \mathcal{U}(\bar{D}_a)$ by fitting polynomials. However, in the data-driven setting, a such extension can be done implicitly during the training. We only need to compute the loss on the original space D_a and discard the extra output from the underlying space $\bar{D}_a$. This continuation technique is universal. According to the Whitney embedding theorem, any m-dimensional manifold can be smoothly embedded into a Euclidean space $\mathbb{R}^{2m}$.

Domain decomposition

For complex topologies, another technique is to decompose the domain into multiple sub-domains such that each of the domains is equipped with a standard topology. We mainly consider 2-dimensional manifolds in our experiments, although the technique is applicable for any dimension. Given any 2-manifold $\mathcal{M}$, we apply the decomposition

$$\mathcal{M} = D_1 \# \cdots \# D_n$$



(a) Elasticity (truth/prediction) (b) Plasticity (truth/prediction)

Figure 9.2: Elasticity (a) and plasticity problems (b) introduced in section 9.4. The comparison is shown between the reference obtained using a traditional solver (left) and the Geo-FNO result (right).

where # is the connect-sum, and each D_i is homeomorphic to either a sphere S^2 or torus T^2. We then train n sub-models $\mathcal{G}_1 \cdots \mathcal{G}_n$, each equipped with a deformation map $\phi_{a_1} \cdots \phi_{a_n}$. This domain decomposition is also universal since any orientable compact 2-dimensional manifold is homeomorphic to a sphere or a n-genus torus, where n-genus torus can be naturally decomposed into n tori. Compared to continuation, the decomposition method does not require raising the dimensionality, but it needs multiple sub-models. A numerical example is included in Section 9.4.

9.4 Numerical Examples

In this section, we study Geo-FNO numerically, and compare the Geo-FNO against other machine-learning models on PDEs with various geometries, including hyper-elastic problem in section 9.4, plastic problem in section 9.4, advection equation on a sphere problem in section 9.4, airfoil problem in section 9.4, and pipe problem in section 9.4. We demonstrate that Geo-FNO can be applied on irregular domains and non-uniform meshes (See figs. 9.2 and 9.8) and show it is more accurate than combining direct interpolation with existing ML-based PDE solvers such as the standard FNO (Li, Kovachki, et al., 2020a) and UNet(Ronneberger, Fischer, and Brox, 2015), as well as mesh-free methods such as Graph neural operators (GNO) (Li, Kovachki, et al., 2020c) and DeepONet (Lu, Jin, and Karniadakis, 2019) (See table 9.1 and table 9.2). Meanwhile, Geo-FNO perseveres the speed of standard FNO. It can accelerate the numerical solvers up to 10^5 times on the airfoil problem (See section 9.4). All experiments are performed on a single Nvidia 3090 GPU. With no special mention, we train all the models with 500 epochs with an initial learning rate of 0.001 which decays by half every 100 epochs. We use the relative L^2 error for both the training and testing. And the standard Adam optimizer is used. The code is available at `https://github.com/neuraloperator/Geo-FNO`.

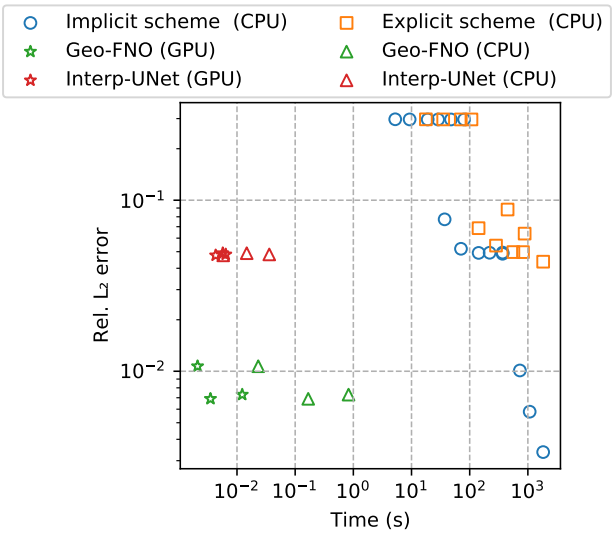


Figure 9.3: The cost-accuracy trade-off for Geo-FNO/UNet and traditional numerical solver with implicit/explicit temporal integrators on the airfoil problem.

Hyper-elastic problem

The governing equation of a solid body can be written as

$$\rho^s \frac{\partial^2 \boldsymbol{u}}{\partial t^2} + \nabla \cdot \boldsymbol{\sigma} = 0$$

where ρ^s is the mass density, $\boldsymbol{u}$ is the displacement vector, $\boldsymbol{\sigma}$ is the stress tensor. Constitutive models, which relate the strain tensor $\boldsymbol{\epsilon}$ to the stress tensor, are required to close the system. We consider the unit cell problem $\Omega = [0, 1] \times [0, 1]$ with an arbitrary shape void at the center, which is depicted in Figure 9.2 (a). The prior of the void radius is $r = 0.2 + \frac{0.2}{1+\exp(\tilde{r})}$ with $\tilde{r} \sim \mathcal{N}(0, 4^2(-\nabla + 3^2)^{-1})$, which embeds the constraint $0.2 \leq r \leq 0.4$. The unit cell is clamped on the bottom edge and tension traction $\boldsymbol{t} = [0, 100]$ is applied on the top edge. The material is the incompressible Rivlin-Saunders material Pascon, 2019 and the constitutive model of the material is given by

$$\boldsymbol{\sigma} = \frac{\partial w(\boldsymbol{\epsilon})}{\partial \boldsymbol{\epsilon}}$$

$$w(\boldsymbol{\epsilon}) = C_1(I_1 - 3) + C_2(I_2 - 3)$$

where $I_1 = \text{tr}(C)$ and $I_2 = \frac{1}{2}[(\text{tr}(C)^2 - \text{tr}(C^2)]$ are scalar invariants of the right Cauchy Green stretch tensor $C = 2\boldsymbol{\epsilon} + \mathcal{I}$. And energy density function parameters are $C_1 = 1.863 \times 10^5$ and $C_1 = 9.79 \times 10^3$. We generate 1000 training data and 200 test data with a finite element solver D. Z. Huang, Xu, et al., 2020 with about 100 quadratic quadrilateral elements. It takes about 5 CPU seconds for each simulation.

The inputs a are given as point clouds with a size of around 1000. The target output is the stress field.

The elasticity problem has an unstructured input format. We compare the mesh-convergent methods such as Geo-FNO, Graph neural operator (GNO), and Deep operator network (DeepONet), as well as interpolation-based ML methods including uniform meshes (FNO, UNet) and heuristic adaptive meshes (R-mesh, O-mesh). The details of these methods are listed below.

- GNO: Since the graph structure is flexible on the problem geometry, graph neural networks have been widely used for complex geometries (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020). In this work, we compare with the graph kernel operator (Li, Kovachki, et al., 2020c; Li, Kovachki, et al., 2020b), which is a graph neural network-based neural operator model. It implements the linear operator $\mathcal{K}$ by message passing on graphs. We build a full graph with edge connection radius $r = 0.2$, width 64, and kernel with 128.

- DeepONet: the deep operator network (Lu, Jin, and Karniadakis, 2019) is a line of works designed with respect to the linear approximation of operator as shown in (T. Chen and H. Chen, 1995). It has two neural networks a trunk net and a branch net to represent the basis and coefficients of the operator. Both DeepONets and neural operators aim to parameterize infinitely dimensional solution operators, but neural operators directly output the full field solution functions, while DeepONets output the solution at specific query points. This design difference gives neural operators, especially FNO, an advantage when the data are fully observed while DeepONet has an advantage when the data are partially observed. In this work, we use five layers for both the trunk net and branch net, each with a width of 256.

- FNO interpolation: as a baseline, we simply interpolate the input point cloud into a 41-by-41 uniform grid and train a plain FNO model (Li, Kovachki, et al., 2020a). As shown in figure 9.4 (c), the interpolation causes an interpolation error, which loses information on the singularities (the red regions). As a result, the testing error is constantly higher than 5%.

- UNet interpolation: Similarly, we train a UNet model (Ronneberger, Fischer, and Brox, 2015) on the interpolated uniform grid. As FNO-interpolation, the error is constantly higher than 5% as shown in the figure 9.4 (d).

- Geo-FNO (R mesh): We consider a heuristic adaptive mesh method for each input geometry by deforming a uniform 41 by 41 grid, as shown in the figure 9.4 (a). The stretching is applied in the radial direction originated at the void center (0.5, 0.5) to attain a finer mesh near the interface of the void. Let r denote the radial distance of mesh points, the deformation map is

$$r \to \begin{cases} r_s + \alpha(r - r_s) + (1-\alpha)\frac{(r-r_s)^2}{r_e - r_s} & r \geq r_s \\ r_s - \alpha(r_s - r) - (1-\alpha)\frac{(r_s-r)^2}{r_s} & r \leq r_s \end{cases}$$

  where r_s and r_e denote the void radius and the unit cell radius in this radial direction, $\alpha = 0.2$ denotes the stretching factor, where the ratio of mesh sizes between these near the void interface and these near the unit cell boundary is about $\frac{\alpha}{2-\alpha}$. It is worth mentioning that the deformation map remains void interface, the origin, and the unit cell boundary. Once the adaptive meshes are generated, we interpolate the input data to the adaptive meshes. The surrogate model is learned on the adaptive meshes. We used four Fourier layers with mode 12 and width 32.

- Geo-FNO (O mesh): Similarly, we design another heuristic adaptive mesh method with cylindrical meshing, as shown in the figure 9.4 (b). A body-fitted O-type mesh is generated for each geometry with 64 points in the azimuth direction and 41 points in the radial direction. The points in the azimuth direction are uniformly located between 0 and 2π, and the points in the radial direction are uniformly located between r_s and r_e, where r_s and r_e denote the void radius and the unit cell radius. Again, we interpolate the input data to the adaptive meshes. The surrogate model is learned on the adaptive meshes. We used four Fourier layers with mode 12 and width 32.

As shown in Table 9.1, for the elasticity problem, the Geo-FNO model has a significantly lower error compared to combining direct interpolation with existing ML-based PDE solvers such as the standard FNO(Li, Kovachki, et al., 2020a) and UNet(Ronneberger, Fischer, and Brox, 2015). Both FNO+interpolation and UNet+interpolation methods have a test error larger that 5%, which is likely caused by the interpolation error. Geo-FNO also has a lower error compared to mesh-free methods such as Graph neural operators (GNO) (Li, Kovachki, et al., 2020c) and DeepONet (Lu, Jin, and Karniadakis, 2019) due to the advantages of the spectral transform, similar to standard FNO (De Hoop et al., 2022). Notice that the Geo-FNO

Model	**m**esh size	**m**odel size	**t**raining time	**t**raining error	**t**esting error
Geo-FNO (learned)	972	1546404	1s	**0.0125**	**0.0229**
GraphNO	972	571617	32s	0.1271	0.1260
DeepONet	972	1025025	45s	0.0528	0.0965
Geo-FNO (R mesh)	1681	1188417	0.6s	0.0308	0.0536
Geo-FNO (O mesh)	1353	1188385	0.5s	0.0344	0.0363
FNO interpolation	1681	1188353	0.5s	0.0314	0.0508
UNet interpolation	1681	7752961	0.9s	0.0089	0.0531

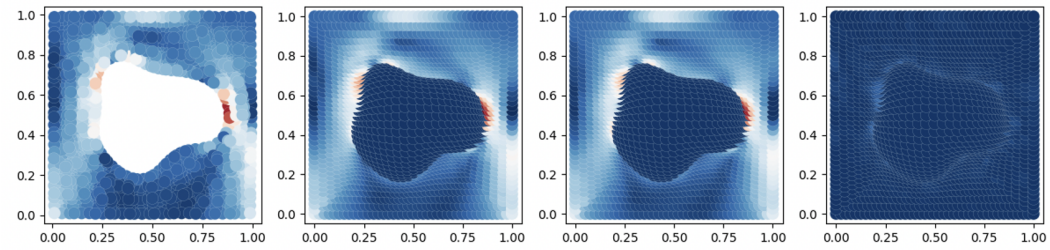
Table 9.1: Benchmark on the elasticity problem. Inputs are point clouds. The table presents the mesh size as the number of input mesh points, the model size as the quantity of training parameters, the training time for each epoch, and the relative training and test errors.

with a learned deformation has better accuracy compared to Geo-FNO with fixed heuristic deformations (R-mesh) and (O-mesh).

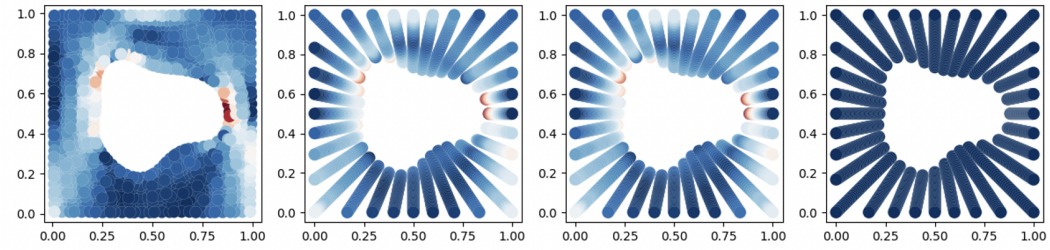
Remark 9.4.1 (Visualization of ϕ_a^{-1}) *Figure 9.5 shows the effects of the coordinate transform, where (a) is the prediction $\mathcal{G}_\theta(a)(x)$ on the original physical mesh and (b) is the prediction on the computational mesh (without transforming back to the physical space) $\mathcal{G}_\theta(a)(\phi_a^{-1}(x))$. The top row visualizes the solution on the original mesh T^i and $\phi_a^{-1}(T^i)$, while the bottom row is the full field solution on T^c directly evaluated on the Fourier coefficients. The solution on the latent space has more "white region", which is latent encoding that does not show up in the final solution. This latent encoding is similar to the Fourier continuation, making the solution easier to be represented with the Fourier basis. As shown in the figure, the solution function on the computational mesh has a cleaner wave structure and is more periodic compared to the physical space, allowing it to be better represented by the Fourier basis. Interestingly, there exists a vertical discontinuity on the latent space around $x = 0.5$ in the latent encoding. Since most features are horizontal, the vertical discontinuity does not affect the output. The gap can be seen as a result of encoding in the high-dimensional channel space.*

Plastic problem

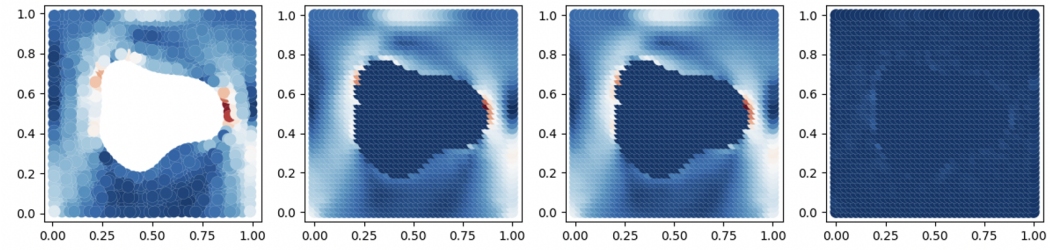
We consider the plastic forging problem where a block of material $\Omega = [0, L]\times[0, H]$ is impacted by a frictionless, rigid die at time $t = 0$. The die is parameterized by an arbitrary function $S_d \in H^1([0, L]; \mathbb{R})$ and traveled at a constant speed v. The block is clamped on the bottom edge and the displacement boundary condition is imposed



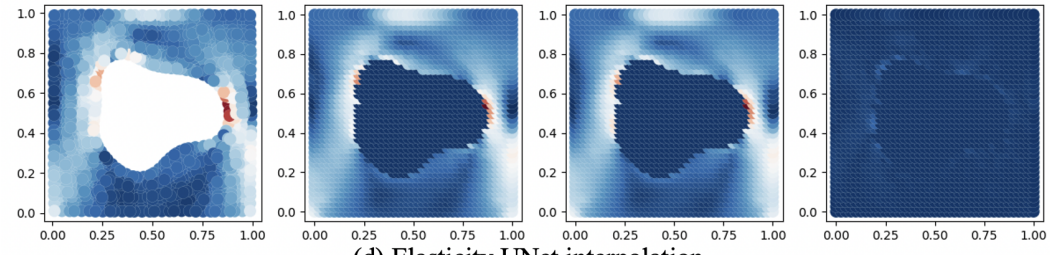
(a) Elasticity R-mesh



(b) Elasticity O-mesh



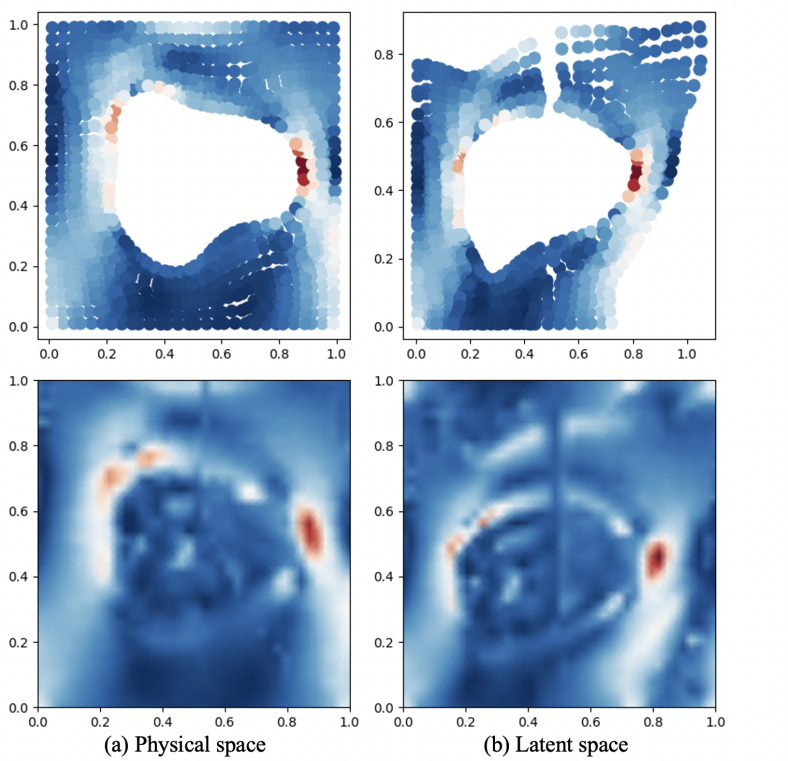
(c) Elasticity FNO interpolation



(d) Elasticity UNet interpolation

The first column is the original unstructured input; the second column is the interpolated input; the third column is the prediction; the last column is the error. As shown in the figures, interpolation causes an error, which is less accurate than geometry-aware methods.

Figure 9.4: Interpolation into different meshes for the elasticity problem



(a) Physical space (b) Latent space

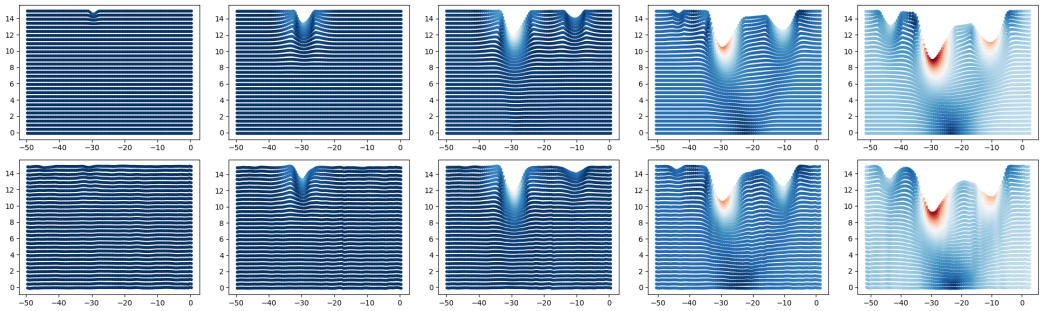
The left column is the prediction of Geo-FNO on the physical space $\mathcal{G}_\theta(a)(x)$; the right column is the prediction of Geo-FNO on the computational space $\mathcal{G}_\theta(a)(\xi) = \mathcal{G}_\theta(a)(\phi_a^{-1}(x))$. The top row is the solution on the input mesh. The bottom row is the full-field solution. The latent space has a cleaner wave structure.

Figure 9.5: Visualization of the deformation map

on the top edge. The governing equation is the same as the previous example but with an elasto-plastic constitutive model given by

$$\begin{aligned}\boldsymbol{\sigma} &= \mathbf{C} : (\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_p)\\ \dot{\boldsymbol{\epsilon}}_p &= \lambda \nabla_{\boldsymbol{\sigma}} f(\boldsymbol{\sigma})\\ f(\boldsymbol{\sigma}) &= \sqrt{\frac{3}{2}} |\boldsymbol{\sigma} - \frac{1}{3}\mathrm{tr}(\boldsymbol{\sigma}) \cdot I|_F - \boldsymbol{\sigma}_Y\end{aligned}$$

where λ is the plastic multiplier constrained by $\lambda \geq 0$, $f(\boldsymbol{\sigma}) \leq 0$, and $\lambda \cdot f(\boldsymbol{\sigma}) = 0$. The isotropic stiffness tensor $\mathbf{C}$ is with Young's modulus $E = 200$ GPa and Poisson's ratio 0.3. The yield strength $\boldsymbol{\sigma}_Y$ is set to 70 MPa with the mass density $\rho^s = 7850\text{kg}\cdot\text{m}^{-3}$. We generate 900 training data and 80 test data by using the commercial Finite Element software ABAQUS Smith, 2009, using 3000 4-node quadrilateral bi-linear elements (CPS4R in ABAQUS's notation). Without lose of generality, we fix $L = 50$mm, $H = 15$mm, and $v = 3$ ms^{-1}. For each sample, we generate a random function S_d by randomly sampling $S_d(x_k)$ on $\{x_k = kL/7; k = 0, .., 7\}$ with



The top row is the truth and the bottom row is the prediction. The five columns represent the changes in time. The color represents the norm of displacement.

Figure 9.6: Geo-FNO on Plasticity

a uniform probability measure. The die geometry is then generated by interpolating $S_d(x_k)$ using piecewise Cubic Hermit Splines. It takes about 600 CPU seconds for each simulation. The target solution operator maps from the shape of the die to the time-dependent mesh grid and deformation. The data is given on the 101×31 structured mesh with 20 time steps.

The plastic forging problem is a time-dependent problem, so we used the FNO3d model as the backbone to do convolution in both spatial dimension and temporal dimensions. Since the data is given on a structured mesh, the deformation (9.16) has an analytical form, so there is no need to learn a deformation. Geo-FNO is equivalent to directly applying the standard FNO on the structured mesh, and hence it perseveres the speed of standard FNO with an inference time of around 0.01 second per. In this experiment, Geo-FNO outputs both the position of the grid as well as the displacement at each grid point. As shown in fig. 9.6, Geo-FNO correctly predicts the evolution of the shape and the displacement. It has a moderate error on the top of the material, which is flat in the ground truth but curved in the prediction. Overall, Geo-FNO serves as an efficient surrogate model with test error 0.0074 (See table 9.2).

Advection equation on unit sphere

We consider the cos bell test introduced in Rasch, 1994, where a cos bell $c(0, x) = h_c(1.0 + \cos(\pi \frac{\text{dist}(x, x_c)}{r}))$ is randomly generated and centered at $x_c = (\lambda_c, \theta_c)$ with radius $r \sim \mathbb{U}[\frac{10\pi}{128}, \frac{20\pi}{128}]$ and height $h_c \sim \mathbb{U}[0.5, 1.5]$ on the unit sphere. Here dist denotes great-circle distance, $\lambda_c, \theta_c \sim \mathbb{U}[\frac{-\pi}{3}, \frac{\pi}{3}]$ denote longitude and latitude. The cos bell is advected following

$$\frac{\partial c}{\partial t} + \nabla(\boldsymbol{v} c) = 0,$$

Model	**A**irfoil		**P**ipe		**P**lasticity	
	training	testing	training	testing	training	testing
Geo-FNO	0.0134	**0.0138**	0.0047	**0.0067**	0.0071	**0.0074**
FNO interpolation	0.0287	0.0421	0.0083	0.0151	–	–
UNet interpolation	0.0039	0.0519	0.0109	0.0182	–	–

The Plasticity requires the mesh as a target output, so interpolation methods do not apply.

Table 9.2: Benchmark on airfoils, pipe flows, and plasticity. Inputs are structured meshes.

Model	**t**opology	**m**esh size	**m**odel size	**t**raining error	**t**esting error
FNO2D	T^2	128x64	2368001	**0.0119**	0.0381
FNO3D	D^3	64x64x64	35397665	> 100%	> 100%
Geo-FNO2D	$D^2 \# D^2$	128x64	4736002	**0.0119**	**0.0332**
UNet2D	D^2	128x64	7752961	0.1964	0.3132
DeepONet	-	128x64	2624769	0.1113	0.1599

Table 9.3: Advection equation on sphere

here the advective speed $\boldsymbol{v}(\lambda, \theta) = \big(\cos\beta + \sin\beta\tan\theta\cos\lambda, -U\sin\beta\sin\lambda\big)$ with $\beta = \pi/2$ and $U = 2\pi/256$. We generate 1000 training data and 200 test data with a finite volume solver. The inputs $c(0, x)$ are given as point clouds with a size of around 8000 on the unit sphere. The target output is the solution $c(t, x)$ at $t = 256/3$.

As shown in Figure 9.7 and table 9.3, Geo-FNO can work on topology differently than torus or disk. Through domain decomposition, particularly by splitting the sphere into the northern and southern hemispheres, we can apply FFT2D for the spherical problem without raising dimensionality. It outperforms baseline models such as the original FNO and UNet models applied on the 2D spherical map projection, whose topology is not a sphere. We also compare against the FNO3D model that embeds the sphere into R^3. It exhibits a remarkably high level of test error, which we attribute to the considerably larger training data and resolution demands associated with 3D learning. This underscores the significance of selecting an appropriate latent computation domain.

Airfoil problem with Euler's equation

We consider the transonic flow over an airfoil, where the governing equation is the Euler equation, as follows,

$$\frac{\partial \rho^f}{\partial t} + \nabla \cdot (\rho^f \boldsymbol{v}) = 0, \qquad \frac{\partial \rho^f \boldsymbol{v}}{\partial t} + \nabla \cdot (\rho^f \boldsymbol{v} \otimes \boldsymbol{v} + p\mathcal{I}) = 0, \qquad \frac{\partial E}{\partial t} + \nabla \cdot \Big((E + p)\boldsymbol{v}\Big) = 0$$

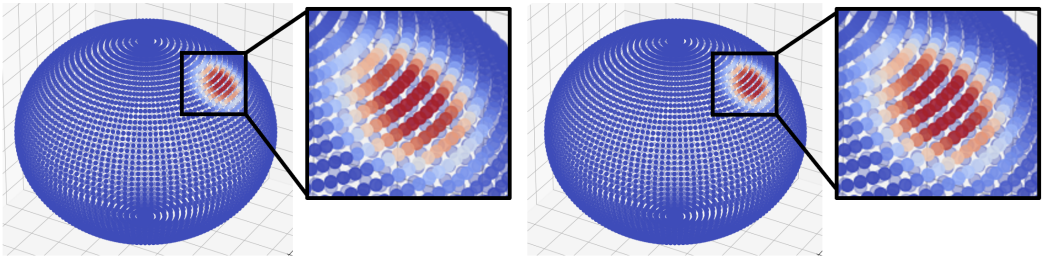


Figure 9.7: Advection equation on the unit sphere. It shows that 2D Geo-FNO with domain decomposition can simulate PDE on general topologies.

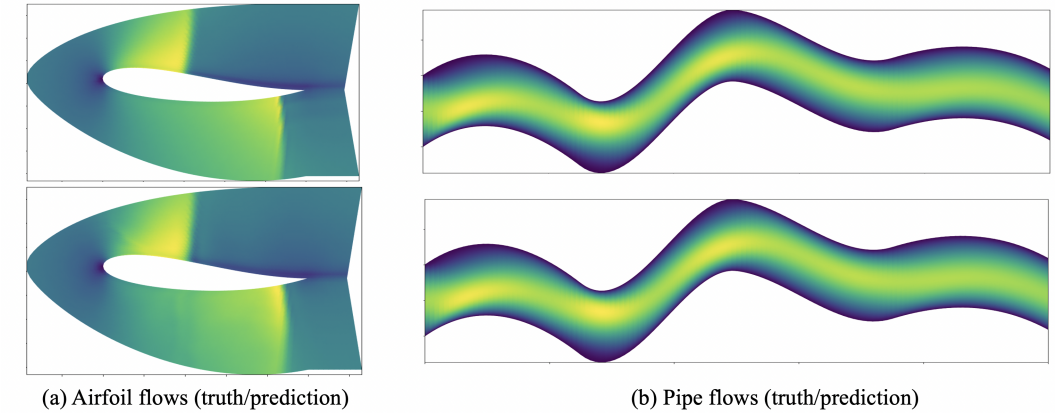


Figure 9.8: The airfoil flows (a) and pipe flows (b) introduced in section 9.4; The comparison is shown between the reference obtained using a traditional solver (top) and the Geo-FNO result (bottom).

where ρ^f is the fluid density, $\boldsymbol{v}$ is the velocity vector, p is the pressure, and E is the total energy. The viscous effect is ignored. The far-field boundary condition is $\rho_\infty = 1, p_\infty = 1.0, M_\infty = 0.8, AoA = 0$ where M_∞ is the Mach number and AoA is the angle of attack, and at the airfoil, no-penetration condition is imposed. The shape parameterization of the airfoil follows the design element approach Farin, 2014. The initial NACA-0012 shape is mapped onto a 'cubic' design element with 8 control nodes, and the initial shape is morphed to a different one following the displacement field of the control nodes of the design element. The displacements of control nodes are restricted to vertical direction only with prior $d \sim \mathbb{U}[-0.05, 0.05]$. We generate 1000 training data and 200 test data with a second-order implicit finite volume solver. The C-grid mesh with about (200×50) quadrilateral elements is used, and the mesh is adapted near the airfoil but not around the shock. It takes about 1 CPU-hour for each simulation. The mesh point locations and Mach number on these mesh points are used as input and output data.

Similar with the plastic problem, the data for the airfoil problem is given on a structured mesh (Omesh), the deformation (9.16) is prescribed, so there is no need to learn a deformation. Geo-FNO is equivalent to directly applying the standard FNO on the structured mesh. The prediction of Geo-FNO on a sample test is presented in fig. 9.8-a, although there are two shocks induced by the airfoil, the Geo-FNO prediction matches well with the reference. We also compare Geo-FNO with the interpolation-based ML methods, which directly interpolates the target on a larger rectangular space. As shown in table 9.2, Geo-FNO outperforms both FNO and UNet with interpolation. In the following discusses, we explore discretization convergence of Geo-FNO model, throughly compare the cost-accuracy trade-off between Geo-FNO and traditional Euler solvers. And finally we demonstrate that we can conduct real-time design optimization with the trained Geo-FNO model.

Discretization Convergence Discretization-convergence is an important property for surrogate models (Kovachki, Li, et al., 2021). It means the model can be applied to any discretizations and resolutions, and further, the same set of parameters can be transferred to different discretizations and resolutions. Such a design philosophy guides the research on neural operators to model the target solution operator as a mapping between function spaces, not just a specific model at one testing resolution. The graph neural operator and Fourier neural operator (with discrete Fourier transform) are both discretization-convergent. However, when implemented with the Fast Fourier transform, Fourier neural operator is restricted to the uniform grids and therefore it loses the discretization-convergent property. The proposed Geo-FNO model, on the other hand, extends FNO (with FFT) to non-uniform meshes, and retains discretization-convergence. Geo-FNO can be trained on a low-resolution dataset and evaluated at a higher resolution. On the Airfoil problem, we train Geo-FNO on a 56×51 mesh. It achieves test errors of $0.0147, 0.0329$, and 0.0428 on 56×51, 111×51, and 221×51 meshes, respectively. Conventional deep learning methods such as U-Nets are not capable of transferring among different resolutions.

Cost-Accuracy Study Geo-FNO is used as the surrogate model to efficiently approximate expensive PDE simulations. Such surrogate modeling is an enabling methodology for many-query computations in science and engineering, e.g., design optimization. In principle, the relative merits of different surrogate models can be evaluated by understanding, for each one, the cost required to achieve a given level of accuracy. De Hoop et. al. demonstrates in (De Hoop et al., 2022), that FNO

has a superior cost-accuracy trade-off among different neural operator approaches. In this section, we study the cost-accuracy trade-off in comparison with traditional numerical solvers with different resolutions. Specifically, we consider the Euler equation airfoil test 9.4, where the reference data are generated by a 220×50 grid with 2000 implicit backward-Euler pseudo-time-stepping iterations. For comparison, we use the same solver but with different spatial resolutions: 220×50, 110×20, and 44×5 grids, and different time integrators, including implicit backward-Euler and explicit Runge-Kutta-2 schemes, with different pseudo time-stepping iterations. For each setup, we estimate the error and CPU time by an average of 10 runs sampled from the data set. The cost-accuracy trade-off is depicted in Figure 9.3. Geo-FNO is at least 10^4x faster while having the same accuracy, for the presented test. We should also mention, the speed-up of Geo-FNO is due to the avoidance of time-stepping iterations, Euler flux computation, and GPU acceleration.

Inverse design Once the Geo-FNO model is trained, it can be used to do the inverse design. We can directly optimize the design parameters to achieve the design goal. For example, as shown in fig. 9.9, the shape of the airfoil is parameterized by the vertical displacements of seven spline nodes. We set the design goal to minimize the drag lift ratio. We first train the model mapping from the input mesh to the output pressure field, then program the maps from the vertical displacement of spline nodes to input mesh and from the output pressure field to the drag lift ratio and finally optimize the vertical displacement of spline nodes in an end-to-end manner. As shown in the figure, the resulting airfoil becomes asymmetry with larger upper camber over the optimization iteration, which increases the lift coefficient and matches the physical intuition. Finally, we use the traditional numerical solver to verify optimized design shape. For the optimized design shape, both Geo-FNO and traditional numerical solver lead to a drag coefficient of 0.04 and a lift coefficient of 0.29.

Pipe problem with Navier-Stokes equation

Finally, we consider the incompressible flow in a pipe, where the governing equation is the incompressible Navier-Stokes equation, as follows,

$$\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v}\nabla)\boldsymbol{v} = -\nabla p + \nu\nabla^2\boldsymbol{v}, \qquad \nabla \cdot \boldsymbol{v} = 0$$

where $\boldsymbol{v}$ is the velocity vector, p is the pressure, and $\nu = 0.005$ is the viscosity. The parabolic velocity profile with maximum velocity $\boldsymbol{v} = [1, 0]$ is imposed at the

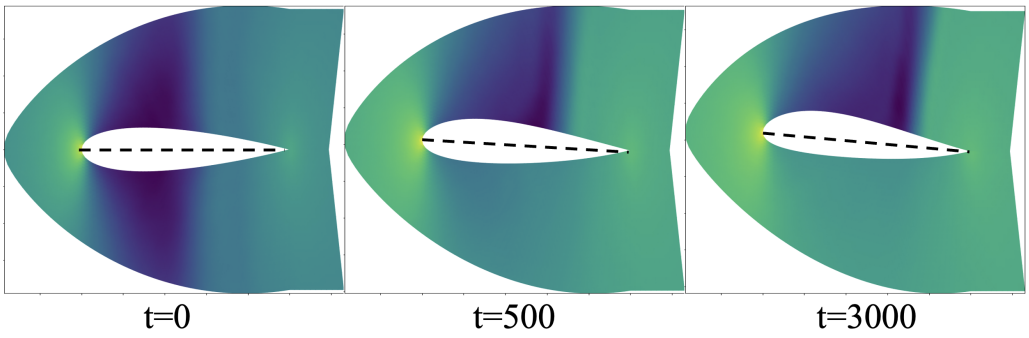


Figure 9.9: The inverse design for the airfoil flow problem with end-to-end optimization. The optimal design using the simulation from the numerical solver matches the prediction from Geo-FNO.

inlet, the free boundary condition is imposed at the outlet, and no-slip boundary condition is imposed at the pipe surface. The pipe is of length 10 and width 1, and the centerline of the pipe is parameterized by 4 piecewise cubic polynomials, which are determined by the vertical positions and slopes on 5 spatially uniform control nodes. The vertical position at these control nodes obeys $d \sim \mathbb{U}[-2, 2]$ and the slop at these control nodes obeys $d \sim \mathbb{U}[-1, 1]$. We generate 1000 training data and 200 test data with an implicit finite element solver with about 4000 Taylor-Hood Q2-Q1 mixed elements D. Z. Huang, Pazner, et al., 2020. It takes about 70 CPU seconds for each simulation. The mesh point locations (129×129) and horizontal velocity on these mesh points are used as input and output data.

Similar with the plastic problem, the data for the pipe problem is given on a structured mesh, the deformation eq. (9.16) has an analytical form, so there is no need to learn a deformation. Geo-FNO is equivalent to directly applying the standard FNO on the structured mesh. The prediction of Geo-FNO on a sample test is presented in fig. 9.8-b, the Geo-FNO is able to capture the boundary layer, and the prediction matches well with the reference. Geo-FNO also outperforms these interpolation-based ML ethods, which directly interpolates the target on a larger rectangular space, as shown in table 9.2.

9.5 Discussion and Conclusion

In this work, we propose a geometry-aware FNO framework (Geo-FNO) that applies to arbitrary geometries and a variety of input formats. The Geo-FNO deforms the irregular input domain into a uniform latent mesh on which the FFT can be applied. Such deformation can be fixed or learned with the FNO architecture in an end-to-end manner. The Geo-FNO combines both the computational efficiency of the FFT

and the flexibility of learned deformations. It is as fast as the original FNO but can represent the variations in solutions and domain shapes more efficiently and accurately. In the end, we also discussed several potential extensions of Geo-FNO.

Physics-informed settings. In this work, we mainly consider learning surrogate models in the data-driven setting. However, the dataset may not always be available. In this case, we can use the physics-informed setting by optimizing the physics-informed equation loss. Given a fixed input a, the output function $u = \mathcal{G}_\theta(a)$ can be explicitly written out using the formula (9.18) and (9.13). The derivative of the deformed basis $\psi_a = e^{2i\pi\langle\phi_a^{-1}(x),k\rangle}$ can be exactly computed using chain rule with the auto-differentiation of the neural network ϕ_a^{-1}. Using the exact gradient, one can minimize the residual error $\mathbb{R}(\mathcal{G}_\theta(a))$ to find out the parameter $\mathcal{G}_\theta(a)$ that represents the solution function. The Geo-PINO method will be an optimization-based spectral method for general geometry.

General topologies. In this work, we mainly studied simple topologies of 2D disks or 2D disks with holes. If the problem topology is more challenging, there does not exist a diffeomorphism from the physical space to the uniform computational space. Thankfully, we can use the Fourier continuation and decomposition to convert the problem domain into simpler ones. It is known that 2D connected orientable surfaces can be classified as either a sphere or an n-genus torus. For spheres, it is natural to use the unit sphere as the computational space and the spherical harmonics as the computational basis. For n-genus torus ($n \geq 2$), usually, there do not exist useful harmonics series, but we can decompose the domain, which requires training multiple FNO models on each of the sub-domain in a coupled manner. We leave the domain decomposition as exciting future work.

Theoretical guarantees. In the end, it will be interesting to extend the universal approximation bound of Fourier neural operator (Kovachki, Lanthaler, and Mishra, 2021) to the solution operator of PDEs defined on general geometries. In (Kovachki, Lanthaler, and Mishra, 2021), the approximation is achieved by using existing pseudo-spectral solvers. Since FNO's architecture can approximate the operation in the pseudo-spectral solvers, FNO can approximate the target solution operators. For general domains, usually, there does not exist a pseudo-spectral solver. However, we can transform the problem into a computational space. By applying the universal approximation bound on the deformed equation $\mathbb{R}^c = 0$, as well as the approximation

bound for the neural network ϕ_a^{-1}, it is possible to obtain a bound for Geo-FNO. We also leave this direction as future work.

References

Allen, Kelsey R et al. (2022). "Physical Design using Differentiable Learned Simulators". In: *arXiv preprint arXiv:2202.00728*.

Babuška, Ivo and Manil Suri (1990). "The p-and hp versions of the finite element method, an overview". In: *Computer methods in applied mechanics and engineering* 80.1-3, pp. 5–26.

Bhatnagar, Saakaar et al. (2019). "Prediction of aerodynamic flow fields using convolutional neural networks". In: *Computational Mechanics*, pp. 1–21.

Bruno, Oscar P, Youngae Han, and Matthew M Pohlman (2007). "Accurate, high-order representation of complex three-dimensional surfaces via Fourier continuation analysis". In: *Journal of computational Physics* 227.2, pp. 1094–1125.

Chen, Tianping and Hong Chen (1995). "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems". In: *IEEE Transactions on Neural Networks* 6.4, pp. 911–917.

De Hoop, Maarten et al. (2022). "The Cost-Accuracy Trade-Off In Operator Learning With Neural Networks". In: *arXiv preprint arXiv:2203.13181*.

Driscoll, Tobin A, Nicholas Hale, and Lloyd N Trefethen (2014). *Chebfun guide*.

Farin, Gerald (2014). *Curves and surfaces for computer-aided geometric design: a practical guide*. Elsevier.

Gottlieb, David and Steven A Orszag (1977). *Numerical analysis of spectral methods: theory and applications*. SIAM.

Huang, Daniel Z, Will Pazner, et al. (2020). "High-order partitioned spectral deferred correction solvers for multiphysics problems". In: *Journal of Computational Physics* 412, p. 109441.

Huang, Daniel Z, Kailai Xu, et al. (2020). "Learning constitutive relations from indirect observations using deep neural networks". In: *Journal of Computational Physics* 416, p. 109491.

Huang, Weizhang and Robert D Russell (2010). *Adaptive moving mesh methods*. Vol. 174. Springer Science & Business Media.

Kovachki, Nikola, Samuel Lanthaler, and Siddhartha Mishra (2021). "On universal approximation and error bounds for Fourier Neural Operators". In: *Journal of Machine Learning Research* 22, Art–No.

Kovachki, Nikola, Zongyi Li, et al. (2021). "Neural operator: Learning maps between function spaces". In: *arXiv preprint arXiv:2108.08481*.

Li, Zongyi, Nikola Kovachki, et al. (2020a). *Fourier Neural Operator for Parametric Partial Differential Equations*. arXiv: 2010.08895 [cs.LG].

– (2020b). *Multipole Graph Neural Operator for Parametric Partial Differential Equations*. arXiv: 2006.09535 [cs.LG].

– (2020c). "Neural operator: Graph kernel network for partial differential equations". In: *arXiv preprint arXiv:2003.03485*.

Li, Zongyi, Hongkai Zheng, et al. (2021). "Physics-informed neural operator for learning partial differential equations". In: *arXiv preprint arXiv:2111.03794*.

Lu, Lu, Pengzhan Jin, and George Em Karniadakis (2019). "DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators". In: *arXiv preprint arXiv:1910.03193*.

Lu, Lu, Xuhui Meng, et al. (2022). "A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data". In: *Computer Methods in Applied Mechanics and Engineering* 393, p. 114778.

Mildenhall, Ben et al. (2020). "Nerf: Representing scenes as neural radiance fields for view synthesis". In: *European conference on computer vision*. Springer, pp. 405–421.

Pascon, João Paulo (2019). "Large deformation analysis of plane-stress hyperelastic problems via triangular membrane finite elements". In: *International Journal of Advanced Structural Engineering* 11.3, pp. 331–350.

Pathak, Jaideep et al. (2022). "FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators". In: *arXiv preprint arXiv:2202.11214*.

Pfaff, Tobias et al. (2020). *Learning Mesh-Based Simulation with Graph Networks*. arXiv: 2010.03409 [cs.LG].

Rasch, Philip J (1994). "Conservative shape-preserving two-dimensional transport on a spherical reduced grid". In: *Monthly weather review* 122.6, pp. 1337–1350.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241.

Sanchez-Gonzalez, Alvaro et al. (2020). "Learning to simulate complex physics with graph networks". In: *arXiv preprint arXiv:2002.09405*.

Sitzmann, Vincent et al. (2020). "Implicit neural representations with periodic activation functions". In: *arXiv preprint arXiv:2006.09661*.

Smith, Michael (2009). *ABAQUS/Standard User's Manual, Version 6.9*. English. United States: Dassault Systèmes Simulia Corp.

Wen, Gege et al. (2022). “U-FNO–An enhanced Fourier neural operator-based deep-learning model for multiphase flow”. In: *Advances in Water Resources*, p. 104180.

Yang, Yan et al. (2021). “Seismic wave propagation and inversion with Neural Operators”. In: *The Seismic Record* 1.3, pp. 126–134.

Zhu, Yinhao and Nicholas Zabaras (2018). “Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification”. In: *Journal of Computational Physics*. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.04.018. URL: http://www.sciencedirect.com/science/article/pii/S0021999118302341.

Chapter 10

GEOMETRY: NEURAL OPERATOR WITH OPTIMAL TRANSPORT

Classical geometry learning problems typically focus on discretized meshes or point clouds. In this work, we extend geometry learning to an operator learning problem by generalizing discretized meshes to mesh density functions. Our approach formulates geometry embedding as an optimal transport (OT) problem, mapping the mesh density function to the uniform density function in reference space. Compared to previous methods with interpolation or shared deformation, our OT-based method has instance-dependent deformation, which is flexible and efficient. Further, for 3D simulations focused on surfaces, our OT-based neural operator embeds the surface geometry into 2D latent space. By constraining computations to the 2D surface manifold, we achieve significant computational efficiency gains compared to volumetric simulation. Experiments utilizing Reynolds-averaged Navier–Stokes equations (RANS) on the ShapeNet-Car and DrivAerNet datasets reveal that our method not only achieves superior accuracy but also significantly reduces computational expenses in terms of both time and memory usage compared to earlier machine learning models. Additionally, our model demonstrates improved accuracy on the FlowBench dataset, underscoring the benefits of utilizing instance-dependent deformation for datasets with highly variable geometries.

10.1 Introduction

Handling complex 3D geometries remains one of the fundamental challenges in scientific computing. While standard numerical solvers based on finite element or spectral methods have been successful on simple regular domains, they struggle with complex geometries due to computationally expensive meshing processes that often require iterative refinement. The challenge of geometric modeling poses a significant obstacle across multiple domains, including fluid dynamics, solid mechanics, and earth science applications. This challenge is particularly evident in 3D aerodynamic simulations of automobiles, where a single shape takes over three hundred hours on CPU (Elrefaie, Dai, and Ahmed, 2024) or ten hours on GPU (Zongyi Li, N. B. Kovachki, et al., 2023).

Machine learning methods have emerged as promising alternatives for solving partial

differential equations (PDEs) on complex geometries, offering dramatic improvements in computational efficiency (Bhatnagar et al., 2019; Pfaff et al., 2021; Thuerey et al., 2020; Hennigh et al., 2021). These approaches can operate effectively at lower resolutions compared to traditional numerical solvers, significantly reducing computational overhead. However, most existing ML-based methods are constrained to specific resolutions, limiting their flexibility and broader applicability. To address this limitation, we focus on neural operators, a recent breakthrough in scientific computing that offers a resolution-independent approach to solving PDEs.

Neural operators for complex geometries. Neural operators represent an innovative class of data-driven models designed to directly learn the mapping of solution operators for PDEs in a mesh-free manner (Zongyi Li, N. Kovachki, et al., 2020a; N. Kovachki et al., 2023; Lu et al., 2021). Unlike conventional deep learning models, neural operators are designed to be invariant to discretization, making them particularly effective for solving PDEs. Significant advances in neural operator research have focused on addressing PDEs with complex geometries (Zongyi Li, D. Z. Huang, et al., 2022; M. Yin et al., 2024; Ahmad et al., 2024), primarily through embedding the geometries into uniform latent spaces where efficient spectral methods such as the Fast Fourier Transform (FFT) (Cooley and Tukey, 1965) can be applied.

The Geometry-Aware Fourier Neural Operator (Geo-FNO) (Zongyi Li, D. Z. Huang, et al., 2022) marked a significant advancement in this direction by introducing a diffeomorphic mapping from physical to computational domains structured as regular grids. This innovation enabled the application of the FFT in latent computational spaces, dramatically improving computational efficiency. However, Geo-FNO faces two major limitations. It learns a shared deformation map for a class of shapes, which cannot address case-dependent geometry features. In addition, GeoFNO encodes and decodes the geometry using a Fourier transform based on computationally expensive matrix-vector multiplication, which restricts scaling to large 3D simulations.

Building on these ideas, the Geometry-Informed Neural Operator (GINO) (Zongyi Li, N. B. Kovachki, et al., 2023) combined Graph Neural Operators (GNO) (Zongyi Li, N. Kovachki, et al., 2020b) with Fourier Neural Operators (FNO) (Zongyi Li, N. Kovachki, et al., 2020a). By leveraging the adaptability of graphs for local interactions and the computational efficiency of FFT for global physics, it became the first neural operator capable of tackling large-scale 3D aerodynamics challenges.

Despite its potential, the method struggles with the inherent limitations of graph embeddings' locality and the high computational cost of 3D latent spaces. These challenges are especially evident in large-scale scenarios.

Another line of works encode the geometry into structure-free tokens based on transformer (Hao et al., 2023; Wu et al., 2024; Alkin et al., 2024) or implicit neural representation (Y. Yin et al., 2022; Serrano et al., 2023; H. Chen et al., 2023; P. Y. Chen et al., 2022). These methods are flexible and generic, but they generally do not preserve geometric properties in their encoding. Meanwhile, their encoders are generally not invertible, which limits their applications in inverse meshing optimization and shape design.

Despite these advances, current neural operator approaches continue to grapple with the computational burden of 3D PDEs and the challenge of learning operators across diverse geometries. To address these challenges, we propose reformulating geometry embedding as an optimal transport problem for each instance. This allows solution operators to be learned directly on the surface manifold, with deformation tailored to each instance. This novel approach fundamentally transforms the handling of complex geometries in neural operators.

Geometry encoding with optimal transport Optimal transport provides a rigorous mathematical framework for determining the most efficient transformation between densitys. We leverage this framework by interpreting surface meshes as continuous density functions, where mesh density reflects the underlying geometric complexity and surface curvature. Our key insight is to formulate the geometry embedding problem as an optimal transport problem that maps these mesh density functions to uniform density functions in a latent space.

Unlike traditional approaches that rely on direct projection and interpolation—which often result in problematic point clustering and density distortions—optimal transport inherently preserves the structural properties of the mesh while ensuring a smooth, physically meaningful transformation, as illustrated in Figure 10.1. This preservation property shares conceptual similarities with adaptive moving mesh methods (Budd, Russell, and Walsh, 2015), but offers flexibility for different topologies. Recent computational advances, particularly the Sinkhorn algorithm (Cuturi, 2013), have made optimal transport practically feasible by providing efficient approximations to the transport problem. Building on these developments, we demonstrate how optimal transport can effectively embed surface mesh sub-manifolds into latent

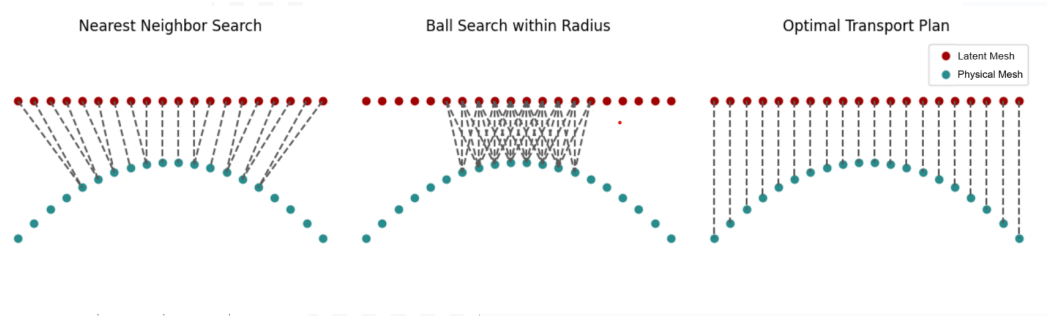


Figure 10.1: OT plans as bi-partite graphs. In the figure, the green nodes represent the input shape and the red nodes represent the latent grid. Compared to other graph mapping strategies such as ball connection and nearest neighbor connection, OT preserves the global measure, which is essential for computing integral operators.

space while maintaining their essential geometric properties, as illustrated in Figure 10.2.

Conformal mapping (Gu et al., 2004; Choi, Lam, and Lui, 2015) represents a common approach for mapping irregular meshes to canonical manifolds. These mappings preserve local angles, making them particularly suitable for Laplacian-type equations. The smoothness and harmonic properties of conformal mappings work exceptionally well with numerical solvers such as finite element methods, as they have less local distortion on each cell within a mesh. However, despite these advantages, conformal mapping is suboptimal for evaluating the integral operators widely employed in neural operators. For this reason, our work adopts optimal transport as the preferred geometric transformation. While optimal transport may not be as smooth locally, they preserve global measure, which is essential for integral operators.

Our Contribution: In this work, we generalize geometry learning from discretized mesh points to mesh density function. Our key innovation lies in formulating geometry embedding as a pre-determined transform that maps the input mesh density function to the uniform density function in the canonical reference space. Such geometric transform is computed as optimal transport.

We explore both the Monge formulation (transport maps) and Kantorovich formulation (transport plans), showing how this unified framework naturally encompasses previous approaches: transport maps generalize the deformation maps in Geo-FNO, while transport plans extend the graph representations in GINO.

Building on this theoretical foundation, we introduce the optimal transport neural

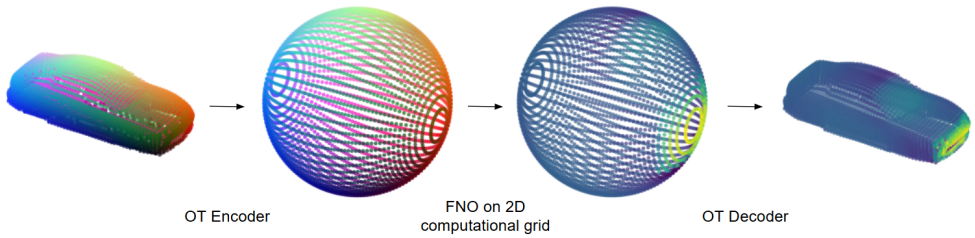


Figure 10.2: Illustration of the optimal transport neural operator (OTNO). (a) OT Encoder: The surface mesh is encoded onto a latent computational mesh, and the OT coupling is visualized by representing the coordinates of points as RGB colors. (b) Fourier Neural Operator (FNO): Within the latent space, we apply S/FNO to calculate solutions on the latent mesh. (c) OT Decoder: We decode the solutions from the latent space back to the original surface mesh; here, the colors indicate solution values.

operator (OTNO), which combines OT-based geometry encoding/decoding with (Spherical) Fourier Neural Operators (Zongyi Li, N. Kovachki, et al., 2020a; Bonev et al., 2023) (Figure 10.2). We use Kantorovich formulation to achieve sparse transport plan and implement it with Sinkhorn algorithm(Cuturi, 2013), and use Monge formulation to obtain bijective map and implement it with Projection pursuit Monge map (PPMM)(Meng et al., 2019).

Our optimal transport technique enables crucial dimension reduction by embedding the surface manifolds from the d-dimensional ambient space into the $(d-1)$-dimensional latent spaces. This capability is particularly valuable for automotive and aerospace applications, where many critical simulations, including Reynolds-averaged Navier-Stokes (RANS) and Large-Eddy Simulation (LES), fundamentally operate on boundary value problems. The input is a 2D surface design, and the desired outputs are surface quantities like pressure and shear velocity that determine total drag.

We validate our method through simulations based on RANS equations on the ShapeNet-Car (Umetani and Bickel, 2018) and DrivAerNet-Car (Elrefaie, Morar, et al., 2024) datasets. Results under different sampling rates (Figure 10.3) show that our approach achieves the fastest convergence rate compared to baseline methods and achieves both the smallest error and shortest time cost when using full dataset. Given different sampling size, our model maintains robust performance in geometry representation and embedding.

Moreover, our OTNO conducts geometry embedding for each shape individually,

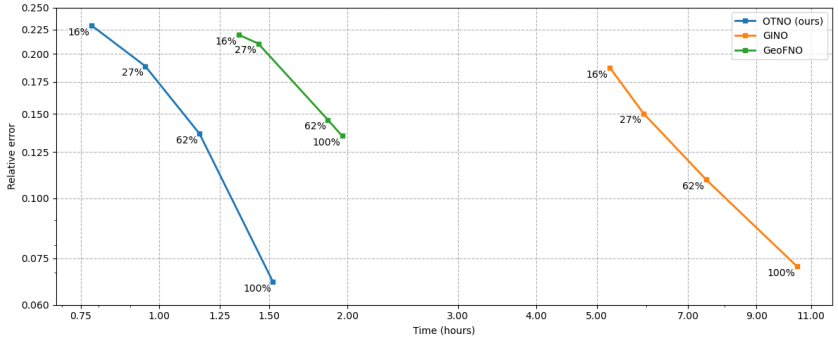


Figure 10.3: Convergence Plot: This figure presents a comparison of convergence rates among different models. We vary the physical mesh size and scale the latent mesh size along the physical size. The data indicates that our model, OTNO, exhibits the fastest exponential convergence rate of 1.85, surpassing both GINO at 1.37 and GeoFNO at 1.32.

distinguishing it from previous methods such as GeoFNO and GINO, which learned a shared deformation network across all geometries. As confirmed by our experiments on the FlowBench dataset, this feature enables our model to better handle datasets composed of a wider variety of shapes. Our main contributions are summarized as follows:

1. A novel optimal transport framework for mesh embedding that unifies and generalizes previous approaches by mapping mesh density functions to latent uniform density functions, bridging the gap between Geo-FNO and GINO methodologies.

2. The Sub-Manifold Method, a dimension-reduction technique for high-dimensional PDEs that restricts solution operators to 2D surface manifolds, implemented with optimal transport and coupled with latent spectral neural operators for efficient PDE resolution in reduced dimensional space, which achieve significant reduction in computational expense.

3. Comprehensive validation on industry-standard datasets—ShapeNet (3.7k points) (Chang et al., 2015) and DrivAerNet (400k points) (Elrefaie, Dai, and Ahmed, 2024)—demonstrates unprecedented efficiency in RANS pressure field prediction. Our method achieves a performance improvement of 2x-8x faster processing and 2x-8x smaller memory usage compared to current machine learning methods, while also slightly enhancing accuracy. Moreover, it is approximately 7,000 times faster than traditional approaches.

4. Optimal transport provides instance-dependent geometry embeddings. Our model notably excels across diverse geometries, as evidenced on the FlowBench dataset, which contains shapes with greater variability.

10.2 Problem Setting: Geometric Operator Learning

In this work, we consider boundary value problems arising from partial differential equations (PDEs). Our aim is to learn the operator from the boundary geometries of PDEs to their boundary solutions. In previous works, the boundary shapes have been parameterized as discrete design parameters (Timmer, 2009), occupancy functions or signed distance functions Zongyi Li, N. B. Kovachki, et al., 2023. In this work, we model the geometries as supports of mesh density functions defined on an ambient Euclidean space. Suppose we have density functions $f \in \mathcal{F}$ defined in domain $\mathbb{R}^d$. Let $\Omega_f = \text{supp}(f)$ for $f \in \mathcal{F}$ be the manifolds in which the PDEs are defined. We assume that $\partial\Omega_f \in V$ for any $f \in \mathcal{F}$, where V is a compact subset of $\mathbb{R}^d$. Denote $\mathcal{L}$ and $\mathcal{B}$ as a partial differential operator and a boundary operator. The primary high-dimensional PDEs are as follows:

$$\begin{aligned} \mathcal{L}(u) &= h, \quad \text{in } \Omega_f, \\ \mathcal{B}(u) &= b, \quad \text{in } \partial\Omega_f, \end{aligned}$$

where $\partial\Omega_f$ denote the boundary manifold of Ω_f, which is a sub-manifold of Ω_f. h and b are some fixed functions on $\mathbb{R}^d$. We assume $(\mathcal{L}, \mathcal{B})$ satisfies that for any (h, b), the PDE (10.1) has a unique solution u on Ω_f. Denote $\mathcal{U}$ as the set of solution functions. Then the solution operator of these high-dimensional PDEs is given by:

$$\begin{aligned} \mathcal{G} : \mathcal{F} &\to \mathcal{U} \\ f &\mapsto u \quad \text{in } \Omega_f, \end{aligned} \tag{10.1}$$

Sub-Manifold Operator For the associated boundary value problems, the solution operator on the sub-manifold $\partial\Omega_f$ is given by:

$$\mathcal{G}^* : f^{\text{sub}} \mapsto u^{\text{sub}} \quad \text{in } \partial\Omega_f, \tag{10.2}$$

where $f^{\text{sub}} = \frac{f|_{\partial\Omega_f}}{\int_{\partial\Omega_f} f(x)dS}$ (dS is a surface measure) denotes the normalized function of f constrained to $\partial\Omega_f$ which is a density function on $\partial\Omega_f$, and u^{sub} denotes our target solution function on $\partial\Omega_f$. Our aim is to solve the operator $\mathcal{G}^*$ in Eq (10.2) on the sub-manifolds $\{\partial\Omega_f : f \in \mathcal{F}\}$.

For linear elliptic PDEs such as the Helmholtz equations on regular domains (Ihlenburg and Babuška, 1995; Hubbert, 1956), solution operators can be explicitly constructed on boundary sub-manifolds through a well-established process: defining basis functions for boundary inputs and solutions via singular value decomposition (SVD), then establishing a linear mapping between these bases. However, this approach breaks down for nonlinear problems like RANS and LES with complex geometries, where explicit linear mappings become mathematically impossible. To overcome this limitation, we introduce a novel approach that combines optimal transport for geometry embedding with neural operators learned directly on the surface sub-manifold, enabling efficient handling of nonlinear boundary value problems.

Reynolds-Averaged Navier-Stokes Equations One common example in computational fluid dynamics is the shape design problem. Given a mesh with mesh density function $f \in V \subset \mathbb{R}^3$, we aim to solve the Reynolds-averaged Navier–Stokes Equations in $\Omega_f = \text{supp}(f)$, which is the open volume outside of automotive or airfoil.

$$\begin{aligned} -\frac{1}{Re}\Delta\overline{\mathbf{v}} + (\overline{\mathbf{v}}\cdot\nabla)\overline{\mathbf{v}} + \nabla\overline{\mathbf{p}} &= \mathbf{h} \quad \text{in } \Omega_f, \\ \nabla\overline{\mathbf{v}} &= 0 \quad \text{in } \Omega_f, \\ \overline{\mathbf{v}} &= \mathbf{b} \quad \text{in } \partial\Omega_f, \end{aligned} \tag{10.3}$$

where $\overline{\mathbf{v}}$ represents the time-averaged velocity field, $\mathbf{b}$ is boundary condition of velocity, and $\overline{\mathbf{p}}$ denotes the time-averaged pressure field. The Re represents the Reynolds number, which accounts for turbulence effects in the averaged flow field. The vector $\mathbf{h}$ includes any external forces acting on the fluid. Given that the boundary condition $\mathbf{b}$ and vector $\mathbf{h}$ are fixed, these equations on various manifolds give rise to the solution operator $\mathcal{G} : f \mapsto u = (\overline{\mathbf{v}}, \overline{\mathbf{p}})$.

Our objective is to solve the pressure field $\overline{\mathbf{p}}$ on the boundary manifold $\partial\Omega_f$ which is the surface of automotive or airfoil in practice. Correspondingly, the solution operator we target is a sub-manifold operator of the original 3D PDE solution operator $\mathcal{G}$:

$$\mathcal{G}^* : f^{\text{sub}} \mapsto u^{\text{sub}} = \overline{\mathbf{p}} \quad \text{in } \Omega. \tag{10.4}$$

where f^{sub} is the density function of the surface mesh of the automotive or airfoil and a mass function on it in practice. Although this setting presents certain limitations, it remains a general framework applicable to many realistic 3D geometry design problems, where the desired solutions are typically concentrated on the surfaces of objects such as cars and airfoils.

10.3 Geometry Embedding as Optimal Transport

In this section, our goal is to embed a complex geometric domain into a simpler latent geometric domain while simultaneously embedding the associated density function. To address this challenge, we construct a computational framework based on optimal transport, as described below.

For convenience, we use f, u and Ω to denote the f^{sub}, u^{sub} and $\partial\Omega_f$ in Eq (10.2). According to the settings from Sec 10.2, f is a density function defined on the complex geometric domain Ω and $\Omega = \text{supp}(f)$. We define a measure μ built from the density f as follows:

$$d\mu = f(x)\,dx \quad \text{on } \Omega. \tag{10.5}$$

The task is then to embed the physical density function f on Ω into a latent density function g on a simple geometric domain Ω^* within the same metric space $\mathbb{R}^d$. This is equivalent to finding a transformation between measures μ and λ, where $d\lambda = g(\xi)d\xi$ represents a uniform measure on a canonical geometric domain Ω^*, such as a unit sphere or torus. And we use x and ξ to represent positions in Ω and Ω^*.

Thereby, we can model the geometries as the density functions (probability measures) and then encode these density functions using transport maps/plans, finally cooperate with the latent operators such as FNO to solve the PDEs, detailed as follows.

Transport Map Given a transport map T from latent measure λ to physical measure μ, which is a function defined on latent domain Ω^* :

$$\begin{aligned} T : (\Omega^*, \lambda) &\to (\Omega, \mu), \\ \xi &\mapsto x, \end{aligned} \tag{10.6}$$

we define the latent neural operator such that maps the transport map T to the latent solution function v on the latent domain Ω^*:

$$\mathcal{G} : T \mapsto v \quad \text{on } \Omega^*. \tag{10.7}$$

The final solution is

$$u(x) = v \circ T^{-1}(x) \quad \forall x \in \Omega. \tag{10.8}$$

Moreover, for any function $n(x)$ on the physical domain Ω, the transport map T enables encoding it onto the latent domain Ω^* through:

$$n^*(\xi) = n(T(\xi)), \quad \forall \xi \in \Omega^*. \tag{10.9}$$

Transport Plan Given a transport Plan P from latent measure λ to physical measure μ, which is a probability measure on $\Omega^* \times \Omega$ with marginals μ on Ω and λ on Ω^*:

$$\begin{aligned} P : (\Omega^*, \lambda) \times (\Omega, \mu) &\to [0, 1], \\ (\xi, x) &\mapsto P(\xi, x), \end{aligned} \tag{10.10}$$

we define the latent neural operator such that maps the marginal map of P to the latent solution function v on the latent domain Ω^*):

$$\mathcal{G} : \frac{\int_\Omega P(\cdot, x) x d\mu(x)}{\int_\Omega P(\cdot, x) d\mu(x)} \mapsto v \quad \text{on } \Omega^*. \tag{10.11}$$

The final solution is

$$u(x) = \int_{\Omega^*} P(\xi, x) v(\xi) d\lambda(\xi) \quad \forall x \in \Omega. \tag{10.12}$$

Moreover, for any function $n(x)$ on the physical domain Ω, the transport plan P enables encoding it onto the latent domain Ω^* through:

$$n^*(\xi) = \int_\Omega P(\xi, x) n(x) d\mu(x). \tag{10.13}$$

Monge Problem: OT Map

Monge originally formulated the OT problem as finding the most economical map to transfer one measure to another (Monge, 1781). Later, Kantorovich introduced a relaxation of these strict transportation maps to more flexible transportation plans, solved using linear programming techniques (Kantorovich, 2006). In the following sections, we explore both types of OT methods—map and plan. Additionally, we analyze state-of-the-art methods for the deformation layer in operator learning, categorizing them into these two types.

The transportation map $T : \Omega^* \to \Omega$ is measure preserving, if for any Borel set $B \subseteq \Omega$,

$$\int_{T^{-1}(B)} d\lambda(\xi) = \int_B d\mu(x), \tag{10.14}$$

denoted as $T_\# \lambda = \mu$.

Monge raised the OT problem: given the transportation cost function $c : \Omega^* \times \Omega \to \mathbb{R}^+$, we aim to find a transportation map $T : \Omega^* \to \Omega$ that minimizes the total transportation cost,

$$\text{(MP)} \quad \inf\{ \int_{\Omega^*} c(\xi, T(\xi)) d\lambda(\xi) \mid T_\# \lambda = \mu \}. \tag{10.15}$$

Moreover, the following theorem and continuity property hold for the Monge formulation.

Theorem 10.3.1 (Existence and uniqueness of transport map (Brenier, 1991)) *Suppose the measures μ and λ are with compact supports $\Omega, \Omega^* \subseteq \mathbb{R}^d$ respectively, and they have equal total mass $\mu(\Omega) = \lambda(\Omega^*)$. Assume the corresponding density functions satisfy $f, g \in L^1(\mathbb{R}^d)$, and the cost function is $c(\xi, x) = \frac{1}{2}|\xi - x|^2$, then the OT map from λ to μ exists and is unique. It can be expressed as $T(\xi) = \xi + \nabla\phi(\xi)$, where $\phi : \Omega^* \to \mathbb{R}$ is a convex function, and ϕ is unique up to adding a constant.*

Lemma 10.3.1 (Continuity of transport map) *Given that cost function is the squared Euclidean distance and λ is a measure with uniform density function with a compact support, if μ is absolutely continuous and strictly positive also with a compact support, then the OT map T is continuous almost everywhere. (This lemma can be easily derived from Theorem 10.3.1.)*

In practice, the measures are discretized on the grids and represented by the discrete measure:

$$\mu(x) = \sum_{i=1}^{n} \mu_i \delta(x - x_i), \qquad \lambda(\xi) = \sum_{i=1}^{n} \lambda_i \delta(\xi - \xi_i), \tag{10.16}$$

where $\mu_i = \frac{f(x_i)}{\sum_{i=1}^{n} f(x_i)}$, $\lambda_i = \frac{1}{n}$, $\mathcal{X} = \{x_i\}$ are the locations of the vertex in complex geometric domain Ω, $\xi = \{\xi_i\}$ are the positions of the computational mesh in Ω^* and δ is the Dirac delta function. Thus $\sum_{i=1}^{n} \mu_i = \sum_{i=1}^{n} \lambda_i$ satisfy measure perserving. The corresponding discrete Monge problem is as follows:

$$\text{(MP)} \quad \min_{T:\xi\to\mathcal{X}} \sum_{i=1}^{n} c(\xi_i, T(\xi_i)) \cdot \lambda_i. \tag{10.17}$$

As for Geo-FNO(Zongyi Li, D. Z. Huang, et al., 2022), the deformation map can be viewed as generalized OT map with a special choice of cost function:

$$\int_{\Omega} c(x, T^{-1}(x)) d\mu(x) := \int_{\Omega} G(T^{-1}(x)) - u^{\text{sub}}(x) d\mu(x) \tag{10.18}$$

where u^{sub} is our target solution function as presented in Eq (10.2). However, this is a non-standard, generalized cost function, since it depends not on $(x, T^{-1}(x))$, but the global solution operator G and domain Ω.

Similar to learning the function ϕ (defined in Theorem 10.3.1) in Monge's formulation, Geo-FNO (Zongyi Li, D. Z. Huang, et al., 2022) implements a skip connection $\xi = \psi(x) + x$ within the deformation network to learn only ψ. The difference is that Geo-FNO adopts an end-to-end approach, optimizing the deformation map based on the final solution error and learning a shared network to implement geometry deformation. In contrast, the Monge problem learns the deformation map separately from the operator learning and optimizing based on the instance-specific transportation cost for each geometry respectively. Despite these differences in optimization strategies, both approaches can be categorized as employing a *map-type* for the deformation layer.

Kantorovich Problem: OT Plan

Relaxing the map constraint, the Kantorovich formulation seeks probability measures P on $\Omega^* \times \Omega$ that attains the infimum,

$$\text{(KP)} \quad \inf\{\int_{\Omega^*\times\Omega} c(\xi, x) dP(\xi, x) \mid P \in \Gamma(\lambda, \mu)\}. \tag{10.19}$$

where $\Gamma(\lambda, \mu)$ denotes the collection of all probability measures on $\Omega^* \times \Omega$ with marginals λ on Ω^* and μ on Ω, and $c : \Omega^* \times \Omega \to \mathbb{R}^+$ is transportation cost function.

For its discrete implementation, we define $\xi = \{\xi_1, \ldots, \xi_{n_1}\}$ and $\mathcal{X} = \{x_1, \ldots, x_{n_2}\}$ as vertex locations in the domains Ω^* and Ω, where ξ is a uniform mesh in computational space. Denote $a = (\lambda(\xi_1), \ldots, \lambda(\xi_{n_1})) = \frac{1}{n_1} 1_{n_1}$ and $b = (\mu(x_1), \ldots, \mu(x_{n_2}))$ as the density mass vectors of the discrete probability measure λ and uniform measure μ, respectively. Define $M \in \mathbb{R}^{n_1 \times n_2}$ as the cost matrix, where each element represents the cost function value $c(\xi_i, x_j)$. The discrete OT problem then seeks to minimize the total transport cost, known as the Wasserstein distance:

$$\begin{aligned} \text{(KP)} \quad W_M(a, b) &= \min_{P\in\Gamma(a,b)} \langle P, M \rangle \\ &= \min_{P\in\Gamma(a,b)} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} M_{ij} \cdot P_{ij} \\ &= \min_{P\in\Gamma(a,b)} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} c(\xi_i, x_j) \cdot P_{ij}, \end{aligned} \tag{10.20}$$

where $\Gamma(a, b)$ represents the set of all feasible coupling matrices, defined as the discrete probability measures $\Gamma(a, b) = \{P \geq 0 \mid P\mathbf{1}_{n_2} = a, P^T\mathbf{1}_{n_1} = b\}$. Moreover, the following sparity property holds for the discrete implementation of the Kantorovich formulation (Peyré, Cuturi, et al., 2019):

Lemma 10.3.2 (Sparsity of Transport Plan) *The solution to the linear programming is sparse; the number of non-zero entries in the transport plan P is at most $n_1 + n_2 - 1$.*

Similar to learning the optimal coupling matrix in the Kantorovich formulation, GINO (Zongyi Li, N. B. Kovachki, et al., 2023) employs a Graph Neural Operator (GNO), which can be interpreted as a learnable Graph Laplacian. GNO constructs the adjacency matrix by performing neighborhood searches within a fixed radius and learns the edge weights using kernel functions. In contrast, the Kantorovich formulation solves a global transport plan by optimizing the Wasserstein distance, which accounts for the pairwise distances between all points across the two domains. Despite these differences in optimization strategies, both approaches can be categorized as employing a *plan-type* transformation layer.

Cost function

We choose the squared Euclidean distance as the cost function in optimal transport due to both its mathematical convenience and its relevance in geometric applications. In the context of transporting probability measures between two geometric domains embedded in 3D space, the squared Euclidean distance $c(x, y) = \|x - y\|^2$ naturally captures the geometric cost of moving mass from one location to another. It emphasizes longer displacements more heavily, encouraging smoother and more localized transport plans. Moreover, it is widely used in the literature and aligns with the assumptions of classical results like Brenier's theorem, which guarantees the existence of a unique optimal map under this cost. Thus, it serves as a principled and standard choice for geometric transformation tasks in 3D Euclid space.

10.4 Optimal Transport Neural Operator (OTNO)

In this paper, we introduce a novel model, the optimal transport neural operator (OTNO), which efficiently integrates optimal transport with neural operators. Our model employs the Projection pursuit Monge map to obtain an approximate solution of the Monge OT map T, and employs Sinkhorn method (Cuturi, 2013) to obtain an approximate solution of the Kantorovich OT plan T. The resulting Map/plan is then utilized to construct the OT encoder/decoder for neural operator. The methodology and implementation are detailed below.

OTNO Algorithm

Given a dataset $\{(\mathcal{X}_j, u_j)\}_{j=1}^N$ of surface sampling meshes and corresponding PDEs' solution values on surface, where $\mathcal{X}_j = \left[x_{j,k}\right]_{k=1}^{n_j^1} \in \mathbb{R}^{n_j^1 \times 3}$ and $u_j = \left[u_{j,k}\right]_{k=1}^{n_j^1} \in \mathbb{R}^{n_j^1 \times s}$. For each $\mathcal{X}_j$, generate a latent surface mesh $\Xi_j = \left[\xi_{j,l}\right]_{l=1}^{n_j^2} \in \mathbb{R}^{n_j^2 \times 3}$ using a 2D parametric mapping from a square grid, where n_j^2 is a perfect square. Compute the normals $\mathcal{N}_j \in \mathbb{R}^{n_j^1 \times 3}$ on $\mathcal{X}_j$ and $\mathcal{H}_j \in \mathbb{R}^{n_j^2 \times 3}$ on Ξ_j.

By solving OT problem from each latent mesh Ξ_j to each physical sampling mesh $\mathcal{X}_j$, we obtain a set of transported mesh $\{\mathcal{X}'_j\}_{j=1}^N$, where $\mathcal{X}'_j = \left[x'_{j,l}\right]_{l=1}^{n_j^2} \in \mathbb{R}^{n_j^2 \times 3}$. Note that $\mathcal{X}'_j$ serves as a representation of the physical surface from which X_j is sampled.

Algorithm 3 Optimal Transport Neural Operator (OTNO)

Given physical mesh $\{\mathcal{X}_j\}_{j=1}^N$, latent mesh $\{\Xi_j\}_{j=1}^N$, transported mesh $\{\mathcal{X}'_j\}_{j=1}^N$ and solution values $\{u_j\}_{j=1}^N$.
Initialize a FNO $\mathcal{G}_\theta$.
for $j = 1$ to N **do**
    1. Build Index Mapping:
        Encoder indices $\mathcal{E} = \left(\arg\min_{k=1,\dots,n_j^1} \|x'_{j,l} - x_{j,k}\|_2 : l = 1, \dots, n_j^2\right)$
        Decoder indices $\mathcal{D} = \left(\arg\min_{l=1,\dots,n_j^2} \|x'_{j,l} - x_{j,k}\|_2 : k = 1, \dots, n_j^1\right)$
    2. OT encoder: $\mathcal{M}_j = \mathcal{X}_j(\mathcal{E}) \in \mathbb{R}^{n_j^2 \times 3}$, where $\mathcal{M}_j$ selects rows from $\mathcal{X}_j$ according to the indices specified in $\mathcal{E}$.
    3. Latent FNO: $v_j = \mathcal{G}_\theta(\mathcal{T}_j)$, where $\mathcal{T}_j = (\Xi_j, \mathcal{M}_j, \mathcal{H}_j \times \mathcal{N}_j(\mathcal{E})) \in \mathbb{R}^{n_j^2 \times 9}$
        ($\mathcal{H}_j \times \mathcal{N}_j(\mathcal{E})$ computes the cross product between rows).
    4. OT decoder: $u'_j = v_j(\mathcal{D}) \in \mathbb{R}^{n_j^1 \times s}$, where u_j selects rows from v_j according to the indices specified in $\mathcal{D}$.
end for
Compute the empirical loss over all dataset instances: $\sum_{j=1}^N \|u'_j - u_j\|_{\mathcal{U}}$.

Encoder For each point $x'_{j,l}$ $(l = 1, \dots, n_j^2)$ in the transported mesh $\mathcal{X}'_j$, we find the closet point in $\mathcal{X}_j$ and denote e_k as the index of this closest point in $\mathcal{X}_j$. Thus, we obtain an index sequence $\mathcal{E} = \left(\arg\min_{k=1,\dots,n_j^1} \|x'_{j,l} - x_{j,k}\| : l = 1, \dots, n_j^2\right) =$

$(e_1, \ldots, e_{n_j^2})$. Using these indices, we encode the mesh $\mathcal{X}_j$ to $\mathcal{M}_j = \mathcal{X}_j(\mathcal{E}) \in \mathbb{R}^{n_j^2 \times 3}$, where $\mathcal{M}_j$ selects rows from $\mathcal{X}_j$ according to the indices specified in $\mathcal{E}$.

Latent Operator In the latent space, we deploy the FNO $\mathcal{G}_\theta$ to execute the latent neural operator $\mathcal{G}$ as introduced in Eq (10.7)(10.11). Denote T as the transport map in Eq (10.7) or marginal map of transport plan in Eq (10.11). Then it can be fundamentally represented by pairs of latent mesh and transported mesh $(\Xi_j, T(\Xi_j)) = (\Xi_j, \mathcal{X}'_j)$. Considering T as a deformation map, we include deformation-related features by using the cross-product over normals $\mathcal{H}_j \times T(\mathcal{H}_j)$ (further discussion on normal feature is in Sec 10.6). Thus, we configure $\mathcal{T}_j$ as $(\Xi_j, \mathcal{X}'_j, \mathcal{H}_j \times T(\mathcal{H}_j))$ to fully encapsulate the map's properties. To enhance the quantity of surface representation, we substitute points in $\mathcal{X}'_j$ with their closest counterparts in $\mathcal{X}_j$, i.e. use $\mathcal{M}_j = \mathcal{X}_j(\mathcal{E})$ to replace $\mathcal{X}'_j$. Similarly, we use $\mathcal{N}_j(\mathcal{E})$ to replace the $T(\mathcal{H}_j)$. Consequently, $\mathcal{T}_j = (\Xi_j, \mathcal{M}_j, \mathcal{H}_j \times \mathcal{N}_j(\mathcal{E}))$ serves as a comprehensive representation of the deformation map T, as well as the input for the latent FNO.

Decoder Corresponding to the encoder, we compute the index d_k of the closest point in the transported mesh $\mathcal{X}'_j$ for each point $x_{j,k}$ $(k = 1, \ldots, n_j^1)$ in $\mathcal{X}_j$ and build a decoder index sequence $\mathcal{D} = \left(\arg \min_{l=1,\ldots,n_j^2} \|x'_{j,l} - x_{j,k}\|_2 : k = 1, \ldots, n_j^1 \right) = (d_1, \ldots, d_{n_j^1})$. Using these indices, we decode the solutions v_j back to physical surface.

Note that the latent surface mesh $\Xi_j \in \mathbb{R}^{n_j^2 \times 3}$ is constructed from a 2D parametric grid. This ensures that the input features $T_j \in \mathbb{R}^{n_j^2 \times 9}$ for the FNO are organized on a 2D grid. Consequently, the FNO computations occur in 2D space, leveraging the structured layout of Ξ_j, rather than directly handling the unstructured 3D point cloud.

OTNO with OT Plan

Transported Mesh

Using the Sinkhorn method (Cuturi, 2013) detailed in the following Sec 10.4, we obtain dense coupling matrices P_j that represent the OT plans from the latent computational mesh Ξ_j to the boundary sampling mesh $\mathcal{X}_j$. How to effectively utilize these dense matrices within a neural operator framework become a problem worth discussing.

Directly applying the dense matrix by multiplying it with the mesh will lead to a lower accuracy of the final predictions because it only approximates the solution plan for the Kantorovich problem (10.20). And saving large-scale dense matrices is costly. Therefore we discuss how to use these optimal coupling matrices more effectively. As the matrix P_j represents a discrete probability measure on $\mathcal{X}_j \times \Xi_j$, a practical approach is to focus on the maximum probability elements, referred to as the "Max" strategy. Additionally, a more effective "Mean" strategy involves replacing each point $x' \in \mathcal{X}'_j = P_j\mathcal{X}_j$ with the nearest point in $\mathcal{X}_j$, significantly reducing approximation ambiguity caused by Sinkhorn method. Further, we can encode/decode by the top-k nearest neighbors in $\mathcal{X}_j/\mathcal{X}'_j$ instead of only find the single nearest one. Details of these different strategies are further discussed in Section 10.6.

Latent Mesh

In practice, we design the computational grid to have more vertices than the boundary sampling mesh to ensure maximal information retention during the encoding and decoding processes. Let α be an expansion factor, and the number of points in latent space is then set to $n_j^2 = \lceil\sqrt{\alpha \times n_j^1}\rceil^2$. This approach implies that the encoder functions as an interpolator, while the decoder acts as a selective querier. A simple illustration of the encoder and decoder processes is shown in Fig.10.4. And detailed ablation studies for latent mesh shape and latent mesh size are in Sec 10.6 and Sec 10.6.

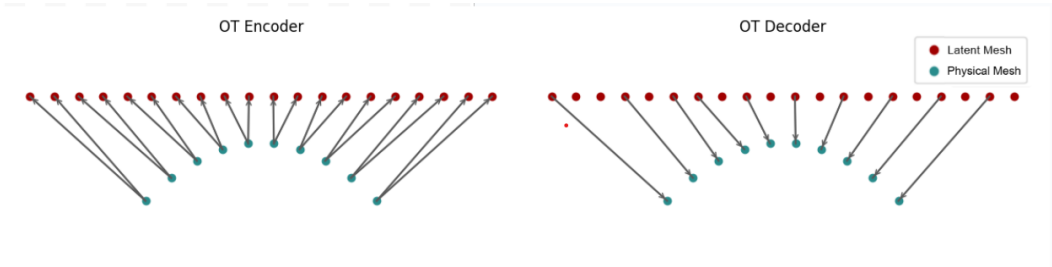


Figure 10.4: Illustration of OT encoder and OT decoder. The curve represents the boundary sampling points and the line denotes the latent computational grid

Voxel Downsampling To achieve a well-balanced input density function, we employ voxel downsampling as a normalization process that constrains the density function within a predefined range $[0, a]$, where a is determined by the voxel size. This approach mitigates excessive clustering in regions with high point density,

such as around the wheels in some some car data, therefore ensuring a more uniform spatial distribution of points.

Sinkhorn Algorithm

Sinkhorn (Cuturi, 2013) method added an entropy regularizer to the Kantorovich potential and greatly improved efficiency. They first propose the Sinkhorn distance that added entropy constraint:

$$d_{M,\alpha}(a,b) = \min_{P \in \Gamma_\alpha(a,b)} \langle P, M \rangle, \tag{10.21}$$

where

$$\Gamma_\alpha(a,b) = \{P \in \Gamma(a,b) \mid \mathbf{KL}(P, ab^T) \leq \alpha\} = \{P \in \Gamma(a,b) \mid h(P) \geq h(a) + h(b) - \alpha\}. \tag{10.22}$$

Then they consider the dual problem that arises from Lagrange multiplier:

$$P^\beta = \underset{P \in \Gamma(a,b)}{\operatorname{argmin}} \quad \langle P, M \rangle - \frac{1}{\beta} h(P), \tag{10.23}$$

This formulation leads to a problem where β adjusts the trade-off between the transport cost and the entropy of the transport plan P. When β increases, the influence of the entropy regularization decreases, making P^β converge closer to the solution of the original Kantorovich problem (10.20). This implies that a larger β leads to a solution that is more accurate and economically efficient.

By introducing the entropy constraint, the Sinkhorn distance not only regularizes the OT problem but also ensures that the solution is computationally feasible even for large-scale problems. This regularization dramatically improves the numerical stability and convergence speed of the algorithm.

Implementation In the implementation, we set $a = \frac{1}{n_1}\mathbf{1}_{n_1}$ given that the sampling mesh $\mathcal{X} = \{x_1, \ldots, x_{n_1}\}$ from the computational fluid dynamics (CFD) simulations typically demonstrates advantages such as increased point density in regions with sharp changes. This uniform mass vector on the mesh $\mathcal{X}$ provides a denser distribution in these critical regions, enhancing aerodynamics prediction. For cases with excessively dense regions, we employ voxel downsampling to prevent excessive density variations, thereby maintaining the feasibility of our uniform mass vector. We set $\beta = 1 \times 10^6$ to ensure that the solution is economical without incurring

excessive computational costs. For computational support, we utilize the geomloss implementation from the Python Optimal Transport (POT) library (Flamary et al., 2021), which supports GPU acceleration and use lazy tensors to store dense coupling matrix obtained from Sinkhorn method.

OTNO with OT Map

Transported Mesh

Using the PPMM, detailed in Section 10.4, we obtain the transported mesh X'_j, which is mapped from the latent mesh Ξ_j to the physical mesh X_j. This results in the same number of points as the latent mesh while representing the distribution of the physical mesh. Since PPMM operates directly on the latent mesh, projecting it towards the physical mesh (as shown in Algorithm 4), the output is the transported mesh itself rather than a mapping that can operate on any function on the latent mesh, and it cannot offer an inverse direction itself. (An OT plan, discretized as a coupling matrix, can handle both.) Therefore, we also compute the indices for encoding and decoding, as outlined in Algorithm 3.

Latent Mesh

Since the latent mesh has the same number of points as the physical mesh, and the 2-dimensional FNO on the latent space requires a square mesh, the number of points in both the latent mesh and the physical mesh should be a perfect square.

Therefore, we first apply voxel downsampling to normalize the mesh density, followed by random sampling to further downsample the mesh so that ensure the number of points is a perfect square. Finally, we generate the latent mesh to match the square number of points in the downsampled physical mesh. We choose a spherical mesh as the latent mesh, given its suitability for the Projection Pursuit Monge Map (PPMM).

Projection pursuit Monge map (PPMM)

The PPMM proposes an estimation method for large-scale OT maps by combining the concepts of projection pursuit regression and sufficient dimension reduction. As summarized in Algorithm 4, in each iteration, the PPMM applies a one-dimensional OT map along the most "informative" projection direction. The direction e_k is considered the most "informative" in the sense that the projected samples $X_k e_k$

Algorithm 4 Projection pursuit Monge map

Input: two matrix $X \in \mathbb{R}^{n\times d}$ and $Y \in \mathbb{R}^{n\times d}$
$k \leftarrow 0$, $X_0 \leftarrow X$
repeat
(a) calculate the most 'informative' projection direction $e_k \in \mathbb{R}^d$ between X_k and Y
(b) find the one-dimensional OT Map $\phi^{(k)}$ that matches $X_k e_k$ to $Y e_k$ (using look-up table)
(c) $X_{k+1} \leftarrow X_k + (\phi^{(k)}(X_k e_k) - X_k e_k^T) e_k^T$ and $k \leftarrow k + 1$
until converge
The final mapping is given by $\hat{\phi} : X \rightarrow X_k$

and Ye_k exhibit the greatest "discrepancy." The specific method for calculating this direction is detailed in Algorithm 1 in paper Meng et al., 2019.

10.5 Experiments

We conducted experiments on three CFD datasets. Two of them are 3D car datasets, where the target prediction is the pressure field or drag coefficient, which depends solely on the car surface—a 2D manifold. The **ShapeNet** dataset includes 611 car designs, each with 3.7k vertices and corresponding average pressure values, following the setup from (Zongyi Li, N. B. Kovachki, et al., 2023). The **DrivAerNet** dataset, sourced from (Elrefaie, Dai, and Ahmed, 2024), contains 4k meshes, each with 200k vertices, along with results from CFD simulations that measure the drag coefficient. Additionally, we further evaluate our model on the 2D **FlowBench** dataset (Tali et al., 2024), which features a wider variety of shapes, including three groups, each containing 1k shapes with a resolution of 512×512. Although the PDE solutions are not on the boundary sub-manifold, preventing our model from reducing dimensions by embedding boundary geometries, this dataset provides an excellent opportunity to explore our model's capabilities across diverse shapes.

Experiments: 3D Car Datasets

ShapeNet Car Dataset

To ensure a fair comparison, we maintained identical experimental settings to those used in GINO paper (Zongyi Li, N. B. Kovachki, et al., 2023). We compared the relative error, total time (including data processing and training), and GPU memory usage against key baselines from (Zongyi Li, N. B. Kovachki, et al., 2023). As detailed in Table.10.1, our method, OTNO, achieved a relative error of **6.70%**,

Model	Relative L2	Total Time (hr)	GPU Memory (MB)
Geo-FNO	13.50%	1.96	12668
UNet	13.14%	6.86	7402
GINO	7.21%	10.45	12734
OTNO(Plan)	**6.70%**	**1.52**	**1890**

Table 10.1: Predict pressure field on ShapeNet Car Dataset

which is a slight improvement over the lowest error reported in the baseline—**7.21%** by GINO. Across all baselines, our method demonstrated reduced total time and lower GPU memory usage. Compared to GINO, our method significantly reduced both time costs and memory expenses by factors of eight and seven, respectively. The visual results for pressure prediction are presented in Fig.10.5a.

DrivAerNet Car Dataset

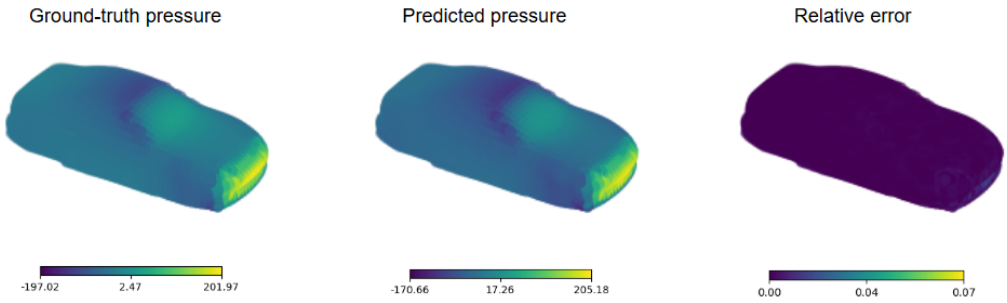
Herein, we follow all the experimental settings from DrivAerNet paper (Elrefaie, Dai, and Ahmed, 2024) and compare our model to RegDGCNN, as proposed in (Elrefaie, Dai, and Ahmed, 2024), and the state-of-the-art neural operator model, GINO (Zongyi Li, N. B. Kovachki, et al., 2023). Note that we do not use pressure data for training; instead, we only use the drag coefficient (Cd). The visual result for Cd prediction is presented in Fig.10.5b. For OTNO, we employ voxel downsampling with a size of 0.05, corresponding to approximately 60k samples (see ablation in 10.6).

As detailed in Table.10.2, OTNO(Plan) demonstrates significantly superior accuracy and reduced computational costs compared to RegDGCNN, halving the MSE, speeding up computations by a factor of 5, and reducing memory usage by a factor of 24. Compared to GINO, OTNO(Plan) achieves a slightly lower MSE, a marginally higher R2 score, and notably reduces total computation time and memory usage by factors of 4 and 5, respectively.

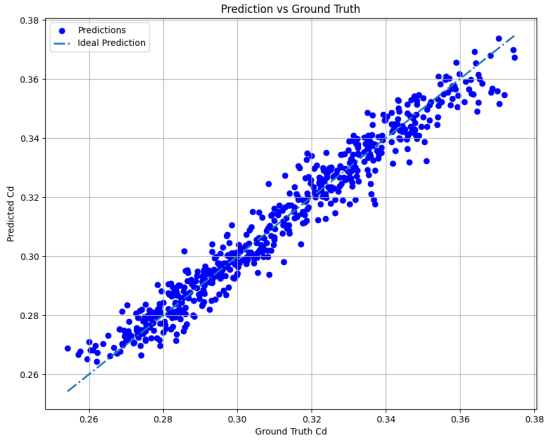
Given that PPMM is designed for large-scale OT maps, we evaluate OTNO(Map) on this large-scale dataset to assess its performance. Unfortunately, both the error and time cost are worse than OTNO(Plan). However, the memory cost is significantly lower. This is primarily because OTNO(Plan) expands the latent space, while OTNO(Map) does not.

Model	MSE (e-05)	R2 Score	Total Time (hr)	GPU Memory (MB)
RegDGCNN	6.63	0.887	10.78	72392
GINO	3.33	0.955	7.73	14696
OTNO(Map)	3.93	0.947	10.63	**2896**
OTNO(Plan)	**3.28**	**0.956**	**5.26**	9702

Table 10.2: Predict drag coefficient (Cd) on DrivAerNet Car Dataset (single A100)



(a) Results of pressure on ShapeNet dataset.



(b) Results of drag coefficient on DrivAerNet dataset

Figure 10.5: Results visualization for OTNO on Car Datasets

Showcase of Dataset with Diverse Geometries - 2D Flow Datasets

To assess the performance of instance-dependent deformation in our model across diverse geometries, we conducted further evaluations using the FlowBench dataset (Tali et al., 2024). We use two metrics for training:

1. M1: *Global metrics*: The errors of in velocity and pressure fields over the

entire domain.

2. M2: *Boundary layer metrics*: The errors in velocity and pressure fields over a narrow region around the object. We define the boundary layer by considering the solution conditioned on the Signed Distance Field ($0 \leq SDF \leq 0.2$).

The results of training using the M1 metric (global) and M2 metric (boundary) are presented in Table 10.3, and Table 10.4, respectively. We present a subset of the visualization results in Fig. 10.6.

The results demonstrate that OTNO significantly outperforms in accuracy under both the M1 (global) and M2 (boundary) metrics, particularly for M1. Furthermore. And OTNO achieves notably better accuracy across all three groups, especially for G2 (harmonics). However, cost reduction is not observed in the FlowBench dataset. It is important to note that the cost reductions seen in car datasets stem from the sub-manifold method, which employs optimal transport to generate 2D representations and perform computations in 2D latent space instead of 3D. In the FlowBench dataset, however, the solutions are not restricted to the boundary sub-manifold. Even for the M2 metric, although the relevant data is close to the boundary with a width of 0.2, it does not reduce to a 1D line. As a result, computations cannot be confined to the sub-manifold, and the associated cost reduction benefits are consequently absent.

We do not present Geo-FNO results on this dataset as the relative L2 errors consistently exceed 60%. Our analysis suggests that Geo-FNO, which uses an end-to-end approach to learn a shared deformation map and the latent operator, is less suited for the diverse shapes present in the FlowBench dataset. In contrast, our OTNO model, which solves the OT plan/map for each shape separately, exhibits superior performance on diverse geometries.

10.6 Ablation Studies

In this section, we discuss ablation experiments for the optimal transport neural operators. We will first cover transport plan formulation, and then transport map based on PPMM.

Group	Model	Relative L2	Time per Epoch (sec)	GPU Memory (MB)
G1	FNO	16.27%	43	4548
	GINO	8.62%	371	57870
	OTNO	**3.06%**	578	26324
G2	FNO	56.67%	43	4548
	GINO	43.16%	390	58970
	OTNO	**7.16%**	603	22868
G3	FNO	23.20%	44	4548
	GINO	13.27%	383	73140
	OTNO	**4.02%**	606	26008

Table 10.3: Prediction under M1 Metric (global) on FlowBench Dataset (single A100)

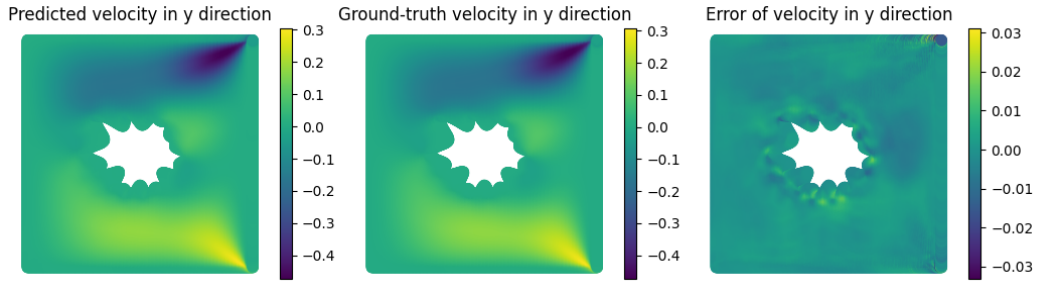
Group	Model	Relative L2	Time per Epoch (sec)	GPU Memory (MB)
G1	FNO	5.65%	43	4536
	GINO	5.83%	190	67632
	OTNO	**3.91%**	135	7300
G2	FNO	29.37%	43	4534
	GINO	19.74%	177	73792
	OTNO	**14.36%**	124	7012
G3	FNO	10.47%	43	4538
	GINO	10.69%	219	67838
	OTNO	**7.18%**	153	11406

Table 10.4: Prediction under M2 Metric (boundary) on FlowBench Dataset

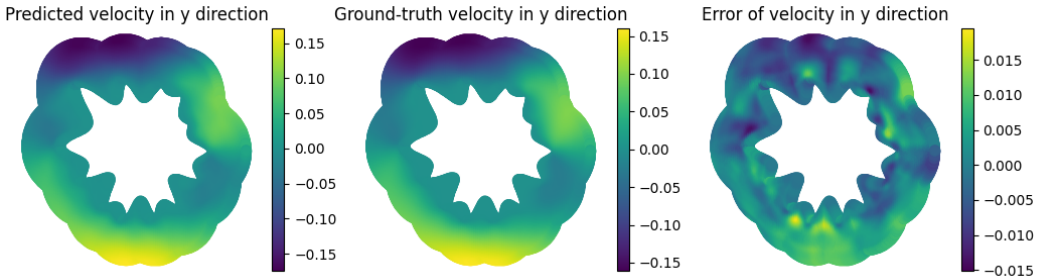
OTNO with OT Plan (Sinkhorn)

Encoder and Decoder Strategy

Using the Sinkhorn algorithm, we solve the Kantorovich optimal transport problem between the latent mesh $\Xi \in \mathbb{R}^{n_2}$ and the physical mesh $\mathcal{X} \in \mathbb{R}^{n_1}$, resulting in a large, dense coupling matrix $P \in \mathbb{R}^{n_2 \times n_1}$ that approximates the OT plan. The direct way to get transported mesh is as $\mathcal{X}' = P \cdot \mathcal{X}$, we refer to as the Matrix Strategy. However, storing these large, dense matrices for each physical mesh is too costly, and the approximate matrices obtained from the Sinkhorn Method introduce a degree of imprecision in the results. Therefore, this section discusses strategies to implement these approximate matrices more efficiently, aiming to conserve memory and reduce imprecision.



(a) Results of velocity in y direction under M1 metric (global).



(b) Results of velocity in y direction under M2 metric (boundary).

Figure 10.6: Results Visualization for OTNO on FlowBench Dataset

1. **Max vs Mean**

   As the matrix $P \in \mathbb{R}^{n_2 \times n_1}$ represents a discrete probability measure on $\Xi \times \mathcal{X}$, a natural approach to take use of the plan is to transport by the maximum probability, termed as "Max" strategy. Let $\mathcal{X} = (x_1, \ldots, x_{n_1}) \in \mathbb{R}^{n_1 \times d}$. Denote $P = (p_{k,l})_{n_2 \times n_1}$. We calculate the indices of the max element in each row

   $$i_k = \arg\min_j \{p_{k,j} : j = 1, \ldots, n_1\}, \text{for } k = 1, \ldots, n_2, \tag{10.24}$$

   and then use these indices to get a refined transported mesh: $\mathcal{M} = (x_{i_1}, \ldots, x_{i_{n_2}}) \in \mathbb{R}^{n_2 \times d}$.

   Another approach, perhaps more intuitive, involves finding the element in mesh $\mathcal{X}$ that is closest to the directly transported mesh $\mathcal{X}' = (x'_1, \ldots, x'_{n_2}) \in \mathbb{R}^{n_2 \times d}$. We name this approach as "Mean" strategy due to the weight product across rows in $\mathcal{X}' = P \cdot \mathcal{X}$. The specific process is described as follows: first compute the indices

   $$i_k = \arg\min_j \{|x'_k - x_j| : j = 1, \ldots, n_1\}, \text{for } k = 1, \ldots, n_2. \tag{10.25}$$

   The refined transported mesh is $\mathcal{M} = (x_{i_1}, \ldots, x_{i_{n_2}}) \in \mathbb{R}^{n_2 \times d}$.

   As shown in Table.10.5, "Mean" strategy has a better performance.

Dataset	Matrix	Max	Mean
ShapeNet (Relative L2)	7.21%	7.01%	**6.70%**
DrivAerNet (MSE e-5)	5.6	4.8	**3.4**

Table 10.5: Comparison of Max vs. Mean Strategy

Dataset	Multi Enc	Multi Dec	Single
ShapeNet (Relative L2)	20.55%	72.30%	**6.70%**
DrivAerNet (MSE e-5)	4.2	4.5	**3.4**

Table 10.6: Comparison of Multi vs. Single Encoder/Decoder Strategy

2. **Single vs Multiple**

   Besides the implementation of OT, integrating OT with FNO poses another significant question. For FNO, the requirement of inputs is just need to be features on a latent grid, suggesting that the encoder's output can combine multiple transported distributions, same for the decoder. Thus, under the "Mean" strategy, we further investigate utilizing indices of the top k closest elements, termed as "Multi-Enc" and "Multi-Dec". We choose $k = 8$ for "Multi-End" and $k = 3$ for "Multi-Dec" setups. The comparative results are presented in Table 10.6, indicating that "Multi-Enc" and "Multi-Dec" configurations underperform relative to "Single" strategy which utilize the index of the closest point instead of top k closest points.

Normal Features

Our model, which incorporates latent FNO, learns the operator mapping from T to the latent solution function v, where T represents the transport map or the marginal map of the transport plan, as described in equations (10.7) and (10.11). The basic representation of the map T in our experiments can be defined as $\mathcal{T} = (\Xi, T(\Xi))$. The previous section discussed how to refine the transported mesh $T(\Xi)$, and in this section, we further explore the addition of normal features to enhance the representation $\mathcal{T}$, leveraging the normal's ability to describe a surface.

We propose three different approaches to integrate normals and compare them with the case where no normal features are added. The three methods are as follows: "Car": Only add car surface normals. "Concatenate": Add the concatenation of latent surface normals and car surface normals as normal features. "Cross Product":

Add the cross product of latent surface normals and car surface normals as normal features to capture the deformation information of the transport.

As shown in Table 10.7, the "Cross Product" method performs the best.

Dataset	Non	Car	Concatenate	Cross Product
ShapeNet (Rel L2)	7.19%	6.82%	6.83%	**6.70%**
DrivAerNet (MSE e-5)	3.4	3.9	3.8	**3.4**

Table 10.7: Comparison of Different Normal Features

Shape of mesh in latent space

We investigated the effects of different shapes for the latent mesh, i.e., the target shapes for optimal transport. The results presented in Table 10.8 indicate that the torus provides the best performance due to its alignment with the periodic Fourier function. Although the capped hemisphere and the spherical surface share the same topological structure as the car surface, their performance is suboptimal. Additionally, we experimented with the double sphere method (Mildenberger and Quellmalz, 2023), which unfortunately yielded worse accuracy compared to the sphere and doubled the time costs.

Dataset	Hemisphere	Plane	Double Sphere	Sphere	Torus
ShapeNet (Rel L2)	8.9%	8.67%	7.41%	7.09%	**6.70%**
DrivAerNet (MSE e-5)	4.7	4.4	4.6	4.1	**3.4**

Table 10.8: Comparison of Different Mesh Shapes in Latent Space

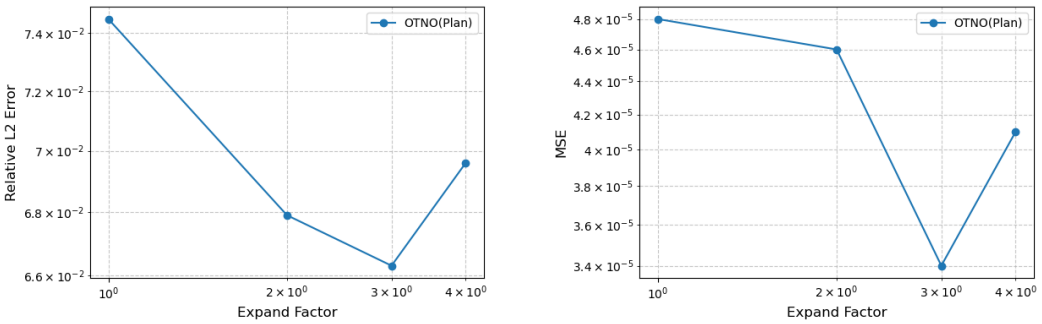
Expand Factor

For OTNO(Plan), we found that expanding the size of the latent mesh within a certain range leads to better results. The ablation experiments for different expansion factors $\alpha = 1, 2, 3, 4$are presented. As shown in Fig. 10.7, on both the DrivAerNet and ShapeNet datasets, $\alpha = 3$ achieves the best accuracy.

OTNO with OT Map (PPMM)

Number of Iterations

The theoretical time complexity of the PPMM is $O(Kn\log(n))$, where K is the number of iterations and n is the number of points. While the original study



(a) Tests of different expand factor for OTNO(Plan) on ShapeNet dataset

(b) Tests of different expand factor for OTNO(Plan) on DrivAerNet dataset

Figure 10.7: Ablation Studies of expand factor for OTNO(Plan)

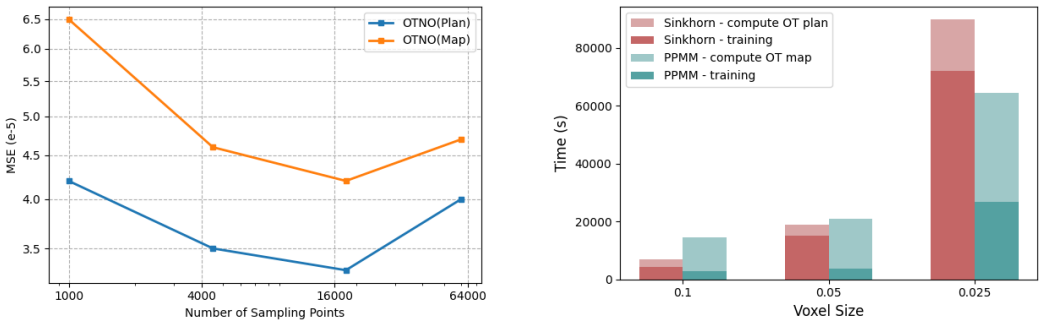
claims that empirically $K = O(d)$ works reasonably well, our experience with 3D car datasets tells a different story. In this section, we conduct ablation studies to explore the relationship between the optimal number of iterations and the number of points. We employ voxel downsampling to generate subsets of the car surface mesh, varying in the number of points. The results for different voxel sizes (r) and iteration numbers (K) are presented in Table 10.9. From these results, we observe that $K \propto r^{-1}$. Given that the car surface mesh is a 2D manifold embedded in 3D space, the number of points n should be inversely proportional to the square of the voxel size r, i.e., $n \propto r^{-2}$. Consequently, $K \propto n^{1/2}$, and the experimental time complexity of PPMM in our model on the car dataset is accordingly $O(n^{3/2}\log(n))$.

Voxel Size / Itr	500	1000	2000	4000
0.2	**6.2**	6.6	6.9	8.0
0.1	5.3	**4.6**	5.5	5.4
0.05	4.8	4.2	**3.9**	4.5

Table 10.9: Results on the DrivAerNet Dataset for Different Voxel Sizes and Different Iteration Numbers of PPMM (MSE (e-5))

Sampling Mesh Size

It is a challenging problem to efficiently and effectively solve large-scale OT problems. We investigate the Sinkhorn method for large-scale OT plans and the PPMM algorithm for large-scale OT maps, and accordingly build two models, OTNO(Plan) and OTNO(Map). In this section, we conduct experiments on the large-scale DrivAerNet dataset to explore our models' ability to handle large sampling meshes. We



(a) Test errors associate with the number of sampling points. We set the voxel size $r = 0.2, 0.1, 0.05, 0.025$, corresponding to about 1k, 4.5k, 18k, 60k points.

(b) Time complexity associate with the number of sampling points. We set the voxel size $r = 0.1, 0.05, 0.025$, corresponding to about 4.5k, 18k, 60k points.

Figure 10.8: Ablation Studies of Sampling Mesh Size for OTNO(Map) and OTNO(Plan) on DrivAerNet Dataset

use voxel downsampling to reduce the mesh with 200k points from the DrivAerNet dataset into a normalized sampling mesh of varying sizes.

As shown in Fig. 10.8a, both OTNO(Plan) and OTNO(Map) achieve their lowest accuracy at a voxel size of 0.05, corresponding to approximately 18k points. But when the sampling mesh size increases more, the accuracy does not improves. This suggests that our model has a range of limitations when dealing with large-scale problems, primarily due to the difficulty of solving large-scale OT problems with high precision. Regarding the comparison between OTNO(Plan) and OTNO(Map), we find that OTNO(Plan) consistently outperforms OTNO(Map) in terms of accuracy, regardless of the sampling mesh size.

From the experimental results, we observe that the training time of OTNO(Plan) is much larger than OTNO(Map) as shown in Fig. 10.8b. This is because we expand the latent space by a factor of 3 for OTNO(Plan), and the FNO on the latent space has linear complexity, attributed to the linear FFT. Note that the time complexity of OTNO(Plan) and OTNO(Map) are $O(n^2)$ and $O(n^{3/2}\log(n))$, respectively, both of which are larger than the training complexity of $O(n)$. Therefore, the overall time complexity of OTNO(Plan) and OTNO(Map) relative to the number of points n should be $O(n^2)$ and $O(n^{3/2}\log(n))$, respectively. These results align with the plots shown in Fig. 10.8b that the time cost of OTNO(Plan) increases faster than that of OTNO(Map) as the sampling mesh size grows.

10.7 Discussion and Conclusion

Recently, several studies have focused on integrating diffeomorphic transformations within PDE solution operators to extract geometric information and ultimately solve PDEs. Examples include DIMON (M. Yin et al., 2024), which employs LDDMM to learn diffeomorphic deformations, and DNO (Zhao et al., 2025), which utilizes harmonic mapping for the same purpose.

A recent work, Diffeomorphic Latent Neural Operators (Ahmad et al., 2024), compared conformal mapping and LDDMM, which are both diffeomorphic mappings, with discrete optimal transport via the Hungarian algorithm, which is a non-diffeomorphic mapping. Their findings suggest that, for invertible spatial transformations, diffeomorphic approaches generally offer better performance.

However, the Hungarian algorithm is designed primarily for discrete assignment problems, which is not suited for spatial transformations between meshes. In contrast, our method employs continuous OT algorithms, which are better equipped for distribution transformations that encode/decode geometric information.

We observe several advantages of our method employing continuous OT algorithms compared to diffeomorphic methods. First, we found that smoothness is an unnecessarily strict requirement—piecewise continuity is often sufficient for effective geometric embeddings. For example, using a sphere surface as a latent shape results in a globally continuous transported mesh, whereas for a torus, the transported mesh is only piecewise continuous. Yet, as shown in Table 10.8, the torus-based representation outperforms the sphere-based one. Secondly invertibility is not always beneficial. Invertibility requires the size of latent mesh should be the same as the size of phyical mesh. However, As illustrated in Fig. 10.7a, expanding the size of latent mesh within a certain range leads to improved performance.

Moreover, the topology of the mapping warrants attention. While diffeomorphisms preserve topology, our OT framework does not require topological consistency. As demonstrated in Table 10.8, the torus representation achieves the best performance despite a change in topology. We also tested this on a 2D elasticity example to assess its generalization capability. The results, shown in Fig. 10.10, confirm that enforcing topological consistency does not yield improved outcomes.

In this work, we propose an OT framework for geometry embedding, which maps density functions from physical geometric domains to latent uniform density functions on regular latent geometric domains. We developed the OTNO model by

Latent grid	Topology	Relative Error
Ring	✔	3.8%
Square	✕	2.7%

Table 10.10: Demonstrate topology independence of OTNO on elasticity

integrating neural operator with optimal transport to solve PDEs on complex geometries. Specifically, there are two implementations: OTNO(Plan) and OTNO(Map), using Kantorovich and Monge formulations, respectively.

Our model achieves particularly good performance in automotive and aerospace applications, where the inputs are 2D surface designs, and the outputs are surface pressure and velocity. These applications provide the opportunity to use optimal transport to embed the physical surface mesh in 3D space into a 2D parameterized latent mesh, allowing computations in a lower-dimensional space. The effectiveness and efficiency of our model in these scenarios are confirmed by our experiments on the ShapeNet-Car and DrivAerNet-Car datasets.

Moreover, by leveraging the advantage of instance-dependent OT Map/Plan, our model handles diverse geometries effectively. As demonstrated on the FlowBench dataset, which includes a wider variety of shapes, OTNO significantly outperforms other models in terms of accuracy.

While our proposed methods demonstrate promise, they present certain limitations that pave the way for future research. A primary challenge lies in the trade-off between computational complexity and accuracy. The Sinkhorn algorithm, underpinning our OTNO(Plan) model, exhibits a complexity of $O(n^2)$, posing scalability challenges for large-scale point clouds. Our alternative, the OTNO(Map) model leveraging the PPMM method, achieves a reduced experimental complexity of $O(n^{3/2}\log(n))$ in a lower-dimensional Euclidean space, yet at the cost of diminished accuracy compared to OTNO(Plan). Consequently, a key direction for future work is the development of novel, sub-quadratic algorithms for optimal transport that can achieve higher accuracy, thereby bridging this performance gap.

A second promising avenue involves a deeper investigation into the optimal selection of the latent sub-manifold. Our experimental results indicate that the optimal latent manifold is non-canonical, diverging from the intuitively favored spherical topology typically chosen for its topological preservation properties. Future efforts will focus on systematically exploring criteria and methods for determining the most suitable

latent manifold structure to enhance model performance.

References

Ahmad, Zan et al. (2024). *Diffeomorphic Latent Neural Operators for Data-Efficient Learning of Solutions to Partial Differential Equations*. arXiv: 2411.18014 [cs.LG]. URL: https://arxiv.org/abs/2411.18014.

Alkin, Benedikt et al. (2024). "Universal physics transformers". In: *arXiv preprint arXiv:2402.12365*.

Bhatnagar, Saakaar et al. (2019). "Prediction of aerodynamic flow fields using convolutional neural networks". In: *Computational Mechanics* 64, pp. 525–545.

Bonev, Boris et al. (2023). "Spherical fourier neural operators: Learning stable dynamics on the sphere". In: *International conference on machine learning*. PMLR, pp. 2806–2823.

Brenier, Yann (1991). "Polar factorization and monotone rearrangement of vector-valued functions". In: *Communications on pure and applied mathematics* 44.4, pp. 375–417.

Budd, CJ, Robert D Russell, and E Walsh (2015). "The geometry of r-adaptive meshes generated using optimal transport methods". In: *Journal of Computational Physics* 282, pp. 113–137.

Chang, Angel X et al. (2015). "ShapeNet: An Information-Rich 3D Model Repository". In: *arXiv:1512.03012 [cs.GR]*.

Chen, Honglin et al. (2023). "Implicit neural spatial representations for time-dependent pdes". In: *International Conference on Machine Learning*. PMLR, pp. 5162–5177.

Chen, Peter Yichen et al. (2022). "Crom: Continuous reduced-order modeling of pdes using implicit neural representations". In: *arXiv preprint arXiv:2206.02607*.

Choi, Pui Tung, Ka Chun Lam, and Lok Ming Lui (2015). "FLASH: Fast landmark aligned spherical harmonic parameterization for genus-0 closed brain surfaces". In: *SIAM Journal on Imaging Sciences* 8.1, pp. 67–94.

Cooley, James W and John W Tukey (1965). "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90, pp. 297–301.

Cuturi, Marco (2013). "Sinkhorn distances: Lightspeed computation of optimal transport". In: *Advances in neural information processing systems* 26.

Elrefaie, Mohamed, Angela Dai, and Faez Ahmed (2024). *DrivAerNet: A Parametric Car Dataset for Data-Driven Aerodynamic Design and Graph-Based Drag Prediction*. arXiv: 2403.08055 [cs.LG]. URL: https://arxiv.org/abs/2403.08055.

Elrefaie, Mohamed, Florin Morar, et al. (2024). "DrivAerNet++: A Large-Scale Multimodal Car Dataset with Computational Fluid Dynamics Simulations and Deep Learning Benchmarks". In: *arXiv preprint arXiv:2406.09624*.

Flamary, Rémi et al. (2021). "POT: Python Optimal Transport". In: *Journal of Machine Learning Research* 22.78, pp. 1–8. URL: http://jmlr.org/papers/v22/20-451.html.

Gu, Xianfeng et al. (2004). "Genus zero surface conformal mapping and its application to brain surface mapping". In: *IEEE transactions on medical imaging* 23.8, pp. 949–958.

Hao, Zhongkai et al. (2023). "Gnot: A general neural operator transformer for operator learning". In: *International Conference on Machine Learning*. PMLR, pp. 12556–12569.

Hennigh, Oliver et al. (2021). "NVIDIA SimNet™: An AI-accelerated multi-physics simulation framework". In: *International conference on computational science*. Springer, pp. 447–461.

Hubbert, M King (1956). "Darcy's law and the field equations of the flow of underground fluids". In: *Transactions of the AIME* 207.01, pp. 222–239.

Ihlenburg, Frank and Ivo Babuška (1995). "Finite element solution of the Helmholtz equation with high wave number Part I: The h-version of the FEM". In: *Computers & Mathematics with Applications* 30.9, pp. 9–37.

Kantorovich, LK (2006). "On a Problem of Monge." In: *Journal of Mathematical Sciences* 133.4.

Kovachki, Nikola et al. (2023). "Neural operator: Learning maps between function spaces with applications to pdes". In: *Journal of Machine Learning Research* 24.89, pp. 1–97.

Li, Zongyi, Daniel Zhengyu Huang, et al. (2022). "Fourier Neural Operator with Learned Deformations for PDEs on General Geometries". In: *ArXiv* abs/2207.05209. URL: http://arxiv.org/abs/2207.05209.

Li, Zongyi, Nikola Kovachki, et al. (2020a). "Fourier neural operator for parametric partial differential equations". In: *arXiv preprint arXiv:2010.08895*.

– (2020b). "Neural operator: Graph kernel network for partial differential equations". In: *arXiv preprint arXiv:2003.03485*.

Li, Zongyi, Nikola Borislavov Kovachki, et al. (2023). "Geometry-Informed Neural Operator for Large-Scale 3D PDEs". In: *NIPS*.

Lu, Lu et al. (2021). "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". In: *Nature machine intelligence* 3.3, pp. 218–229.

Meng, Cheng et al. (2019). "Large-scale optimal transport map estimation using projection pursuit". In: *Advances in Neural Information Processing Systems* 32.

Mildenberger, Sophie and Michael Quellmalz (2023). “A double Fourier sphere method for d-dimensional manifolds”. In: *Sampling Theory, Signal Processing, and Data Analysis* 21.2, p. 23.

Monge, Gaspard (1781). “Mémoire sur la théorie des déblais et des remblais”. In: *Mem. Math. Phys. Acad. Royale Sci.*, pp. 666–704.

Peyré, Gabriel, Marco Cuturi, et al. (2019). “Computational optimal transport: With applications to data science”. In: *Foundations and Trends® in Machine Learning* 11.5-6, pp. 355–607.

Pfaff, Tobias et al. (2021). “Learning mesh-based simulation with graph networks”. In: *ICLR*.

Serrano, Louis et al. (2023). “Operator learning with neural fields: Tackling pdes on general geometries”. In: *Advances in Neural Information Processing Systems* 36, pp. 70581–70611.

Tali, Ronak et al. (2024). “FlowBench: A Large Scale Benchmark for Flow Simulation over Complex Geometries”. In.

Thuerey, Nils et al. (2020). “Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows”. In: *AIAA Journal* 58.1, pp. 25–36.

Timmer, W (2009). “An overview of NACA 6-digit airfoil series characteristics with reference to airfoils for large wind turbine blades”. In: *47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*, p. 268.

Umetani, Nobuyuki and Bernd Bickel (2018). “Learning three-dimensional flow for interactive aerodynamic design”. In: *ACM Transactions on Graphics (TOG)* 37.4, pp. 1–10.

Wu, Haixu et al. (2024). “Transolver: A fast transformer solver for pdes on general geometries”. In: *arXiv preprint arXiv:2402.02366*.

Yin, Minglang et al. (2024). “Dimon: Learning solution operators of partial differential equations on a diffeomorphic family of domains”. In: *arXiv preprint arXiv:2402.07250*.

Yin, Yuan et al. (2022). “Continuous pde dynamics forecasting with implicit neural representations”. In: *arXiv preprint arXiv:2209.14855*.

Zhao, Zhiwei et al. (2025). “Diffeomorphism neural operator for various domains and parameters of partial differential equations”. In: *Communications Physics* 8.1, p. 15.

Chapter 11

APPLICATION: WEATHER FORECAST

FourCastNet, short for *Four*ier Fore*Cast*ing Neural *Net*work, is a global data-driven weather forecasting model that provides accurate short to medium-range global predictions at 0.25° resolution. FourCastNet accurately forecasts high-resolution, fast-timescale variables such as the surface wind speed, precipitation, and atmospheric water vapor. It has important implications for planning wind energy resources, predicting extreme weather events such as tropical cyclones, extra-tropical cyclones, and atmospheric rivers. FourCastNet matches the forecasting accuracy of the ECMWF Integrated Forecasting System (IFS), a state-of-the-art Numerical Weather Prediction (NWP) model, at short lead times for large-scale variables, while outperforming IFS for variables with complex fine-scale structure, including precipitation. FourCastNet generates a week-long forecast in less than 2 seconds, orders of magnitude faster than IFS. The speed of FourCastNet enables the creation of rapid and inexpensive large-ensemble forecasts with thousands of ensemble-members for improving probabilistic forecasting. We discuss how data-driven deep learning models such as FourCastNet are a valuable addition to the meteorology toolkit to aid and augment NWP models.

11.1 Introduction

The beginnings of modern numerical weather prediction (NWP) can be traced to the 1920s. Now ubiquitous, they contribute to economic planning in key sectors such as transport, logistics, agriculture, and energy production. Accurate weather forecasts have saved countless human lives by providing advance notice of extreme events. The quality of weather forecasts has been steadily improving over the past decades (c.f. Bauer, Thorpe, and Brunet (2015) and Alley, Emanuel, and Zhang (2019)). The earliest dynamically-modeled numerical weather forecast for a single point was computed using a slide rule and table of logarithms by Lewis Fry Richardson in 1922 (Richardson, 2007) and took six weeks to compute a 6-hour forecast of the atmosphere. By the 1950s, early electronic computers greatly improved the speed of forecasting, allowing operational forecasts to be calculated fast enough to be useful for future prediction. In addition to better computing capabilities, improvements in weather forecasting have been achieved through better parameterization of fine-

scale processes through deeper understanding of their physics and higher-quality atmospheric observations. The latter has resulted in improved model initializations via data assimilation.

There is now increasing interest around developing data-driven DL-based models for weather forecasting owing to their orders of magnitude lower computational cost as compared to state-of-the-art NWP models (Schultz et al., 2021; Balaji, 2021; Irrgang et al., 2021; Reichstein et al., 2019). Many studies have attempted to build data-driven models for forecasting the large-scale circulation of the atmosphere, either trained on climate model outputs, general circulation models (GCM) (Scher and Messori, 2018; Scher and Messori, 2019; Ashesh Chattopadhyay, Nabizadeh, and Hassanzadeh, 2020), reanalysis products (Weyn, Durran, and Caruana, 2019; Weyn, Durran, and Caruana, 2020; Weyn, Durran, Caruana, and Cresswell-Clay, 2021; Rasp, Dueben, et al., 2020; Rasp and Thuerey, 2021a; Rasp and Thuerey, 2020; Ashesh Chattopadhyay, Mustafa Mustafa, Hassanzadeh, Bach, et al., 2021; Arcomano, Szunyogh, Pathak, et al., 2020; Chantry et al., 2021; Grönquist et al., 2021), or a blend of climate model outputs and reanalysis products (Rasp and Thuerey, 2021a).

Data-driven models have great potential to improve weather predictions by overcoming model biases present in NWP models and by enabling the generation of large ensembles at low computational cost for probabilistic forecasting and data assimilation. By training on reanalysis data or observations, data-driven models can avoid limitations that exist in NWP models (Schultz et al., 2021; Balaji, 2021), such as biases in convection parameterization schemes that strongly affect precipitation forecasts. Once trained, data-driven models are orders of magnitude faster than traditional NWP models in generating forecasts via inference, thus enabling the generation of very large ensembles (Ashesh Chattopadhyay, Mustafa Mustafa, Hassanzadeh, Bach, et al., 2021; Weyn, Durran, Caruana, and Cresswell-Clay, 2021).

In this regard, Weyn, Durran, Caruana, and Cresswell-Clay (2021) have shown that large data-driven ensembles improve subseasonal-to-seasonal (S2S) forecasts over operational NWP models that can only incorporate a small number of ensemble members. Furthermore, a large ensemble helps improve data-driven predictions of extreme weather events in short- and long-term forecasts (Ashesh Chattopadhyay, Nabizadeh, and Hassanzadeh, 2020).

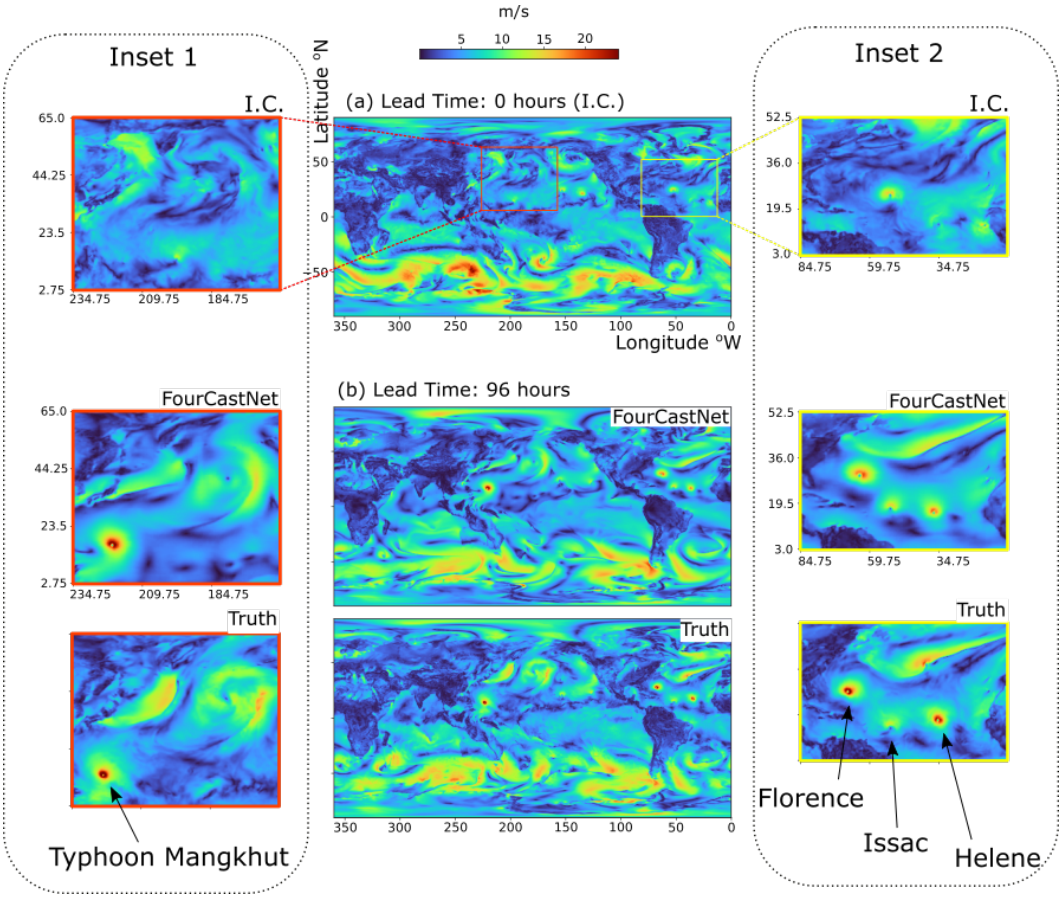
Most data-driven weather models, however, use low-resolution data for training,

usually at the 5.625° resolution as in Rasp and Thuerey (2021b) or 2° as in Weyn, Durran, and Caruana (2020). These prior attempts have achieved good results on forecasting some of the coarse, low-resolution atmospheric variables. However, the coarsening procedure leads to the loss of crucial, fine-scale physical information. For data-driven models to be truly impactful, it is essential that they generate forecasts at the same or greater resolution than current state-of-the-art numerical weather models, which are run at $\approx 0.1°$ resolution. Forecasts at 5.625° spatial resolution, for instance, result in a mere 32×64 pixels grid representing the entire globe. Such a forecast is not able to resolve features smaller than ≈ 500 km. Such coarse forecasts fail to account for the important effects of small-scale dynamics on the large scales and the impact of topographic features such as mountain ranges and lakes on small-scale dynamics. This limits the practical utility of low-resolution forecasts. While low-resolution forecasts may be justified for variables that do not possess complex fine-scale structure, such as the geopotential height at 500 hPa (Z_{500}), higher-resolution data (e.g., at 0.25° resolution) can substantially improve the predictions of data-driven models for variables like low-level winds (U_{10} and V_{10}) that have complex fine-scale structures. Moreover, high-resolution models can resolve the formation and dynamics of high-impact extreme events such as tropical cyclones, which would otherwise be inadequately represented on a coarser grid.

Our approach. We develop FourCastNet, a Fourier-based neural network forecasting model, to generate global data-driven forecasts of key atmospheric variables at a resolution of 0.25°, which corresponds to a spatial resolution of roughly 30 km $\times$ 30 km near the equator and a global grid size of 720×1440 pixels. This allows us, for the first time, to make a direct comparison with the high-resolution Integrated Forecasting System (IFS) model of the European Center for Medium-Range Weather Forecasting (ECMWF).

Figure 11.1 shows an illustrative global near-surface wind speed forecast at a 96-hour lead time generated using FourCastNet. We highlight key high-resolution details that are resolved and accurately tracked by our forecast, including Super Typhoon Mangkhut and three named cyclones heading towards the eastern coast of the United States (Florence, Issac, and Helene).

FourCastNet is about 45,000 times faster than traditional NWP models on a node-hour basis. This orders of magnitude speedup, along with the unprecedented accuracy of FourCastNet at high resolution, enables inexpensive generation of extremely



To prepare this figure, we initialized FourCastNet with an initial condition from the out-of-sample test dataset with the calendar timestamp September 8, 2018 at 00:00 UTC. Starting from this initial condition, the model was allowed to run freely for 16 time-steps of six hours each in inference mode (Figure 11.2(c)) corresponding to a 96-hour forecast. Panel (a) shows the wind speed at model initialization. Panel (b) shows the model forecasts at forecast lead time of 96 hours (upper panel) and the corresponding true wind speeds at that time (lower panel). FourCastNet is able to forecast the wind speeds 96 hours in advance with remarkable fidelity and correct fine-scale features. The forecast accurately captures the formation and track of Super Typhoon Mangkhut that begins to form at roughly $10^{\circ}N$, $210^{\circ}W$ (see Inset 1). Further, the model captures the intensification and track of the typhoon over a period of four days. During the period of this forecast, the model reveals three named hurricanes (Florence, Issac, and Helene) forming in the Atlantic Ocean and approaching the eastern coast of North America.

Figure 11.1: Illustrative example of a global near-surface wind forecast generated by FourCastNet over the entire globe at a resolution of 0.25°.

large ensemble forecasts. This dramatically improves probabilistic weather forecasting. Massive large-ensemble forecasts of events such as hurricanes, atmospheric rivers, and extreme precipitation can be generated in seconds using FourCastNet. This could lead to better-informed disaster response. Furthermore, FourCastNet's reliable, rapid, and cheap forecasts of near-surface wind speeds can improve wind energy resource planning at onshore and offshore wind farms. The energy required to train FourCastNet is approximately equal to the energy required to generate a 10-day forecast with 50 ensemble members using the IFS model. Once trained, however, FourCastNet uses about 12,000 times less energy to generate a forecast than the IFS model. We expect FourCastNet to be only trained once; the energy consumption of subsequent fine tuning is negligible.

FourCastNet uses a Fourier transform-based token-mixing scheme (Guibas et al., 2022) with a vision transformer (ViT) backbone (Dosovitskiy et al., 2021). This approach is based on the recent Fourier neural operator that learns in a resolution-invariant manner and has shown success in modeling challenging partial differential equations (PDE) such as fluid dynamics (Li, Kovachki, et al., 2021). We chose a ViT backbone since it is capable of modeling long-range dependencies well. Combining ViT with Fourier-based token mixing yields a state-of-the-art high-resolution model that resolves fine-grained features and scales well with resolution and size of dataset. This approach enables training high-fidelity data-driven models at truly unprecedented resolution.[1]

In summary, FourCastNet makes four significant contributions to data-driven weather forecasting:

1. FourCastNet predicts, with unparalleled accuracy at forecast lead times of up to one week, challenging variables with complex fine-scale structure, such as surface winds and precipitation. Until now, no deep learning model was able to forecast surface winds on global scales. Additionally, DL models for precipitation on global scales have been inadequate for resolving fine-scale structures. This has important implications for disaster mitigation and wind energy resource planning.

2. FourCastNet, at 0.25° resolution, has eight times greater resolution than state-of-the-art DL-based global weather models. Due to its high resolution and

[1]We estimate that FourCastNet could be trained on currently available GPU hardware in about two months with 40 years of global 5-km data, if such data were available.

accuracy, FourCastNet accurately resolves extreme weather patterns such as tropical cyclones and atmospheric rivers that have been inadequately represented by prior DL models owing to their coarser grids.

3. FourCastNet's predictions are comparable to the IFS model on metrics of Root Mean Squared Error (RMSE) and Anomaly Correlation Coefficient (ACC) at lead times of up to three days. After, predictions of all modeled variables lag close behind IFS at lead times of up to a week. Whereas the IFS model has been developed over decades, contains greater than 150 variables at more than 50 vertical levels in the atmosphere, and is guided by physics, FourCastNet models 20 variables at five vertical levels, and is purely data driven. This comparison points to the enormous potential of data-driven modeling in complementing and eventually replacing NWP.

4. FourCastNet's reliable, rapid, and computationally inexpensive forecasts facilitate the generation of very large ensembles, thus enabling estimation of well-calibrated and constrained uncertainties in extremes with higher confidence than current NWP ensembles that have at most 50 members owing to their high computational cost. Fast generation of 1,000-member ensembles dramatically changes what is possible in probabilistic weather forecasting, including improving reliability of early warnings of extreme weather events and enabling rapid assessment of their impacts.

11.2 Training Methods

The ECMWF provides a publicly available, comprehensive dataset called ERA5 (Hersbach et al., 2020) which consists of hourly estimates of several atmospheric variables at a latitude and longitude resolution of 0.25° from the surface of the earth to roughly 100 km altitude from 1979 to the present day. ERA5 is an atmospheric reanalysis (Kalnay et al., 1996) dataset and is the result of an optimal combination of observations from various measurement sources and the output of a numerical model using a Bayesian estimation process called data-assimilation (Kalnay, 2003). The dataset is essentially a reconstruction of the optimal estimate of the observed history of the Earth's atmosphere. We use the ERA5 dataset to train FourCastNet. While the ERA5 dataset has several prognostic variables available at 37 vertical levels with an hourly resolution, computational and data limitations along with other operational considerations for DL models restricts our choice, based on physical reasoning, to a subset of these available variables to train our model on.

In this work, we focus on forecasting two important and challenging atmospheric variables, namely, (1) the wind velocities at a distance of 10m from the surface of the earth and (2) the 6-hourly total precipitation. There are a few reasons for our focus on these variables. First, surface wind velocities and precipitation require high-resolution models to resolve and forecast accurately because they contain and are influenced by many fine-scale features. Due to computational and model architectural limitations, previous efforts in DL-based weather prediction have not been able to produce global forecasts for these variables at full ERA5 resolution. Near-surface wind velocity forecasts have a tremendous amount of utility due to their key role in planning energy storage, grid transmission, and other operational considerations at on-shore and off-shore wind farms. As we show in Section 11.3, near-surface wind forecasts (along with wind forecasts above the atmospheric boundary layer) can help track extreme wind events such as hurricanes and can be used for disaster preparedness. Our second focus is on forecasting total precipitation where DL models can potentially show great promise. NWP models, such as the operational IFS, have several parameterization schemes to tractably forecast precipitation and since neural networks are known to have impressive capabilities at deducing parameterizations from high-resolution observational data, they are well-suited for this task.

Although we focus on forecasting near-surface wind-speed and precipitation, our model also forecasts with remarkable accuracy several other variables. In our forecast, we include the geopotential height, temperature, wind velocity, and relative humidity at a few different vertical levels, a few near-surface variables such as surface pressure and mean sea-level pressure as well as the integrated total column of water vapor.

FourCastNet: Model Description

To produce our high-resolution forecasts, we choose the Adaptive Fourier Neural Operator (AFNO) model (Guibas et al., 2022). This particular neural network architecture is appealing as it is specifically designed for *high-resolution* inputs and synthesizes several key recent advances in DL into one model. Namely, it combines the Fourier Neural Operator (FNO) learning approach of Li, Kovachki, et al., 2021, which has been shown to perform well in modeling challenging PDE systems, with a powerful ViT backbone.

The vision transformer (ViT) architecture and its variants have emerged as the state-of-the-art in computer vision over the previous years, showing remarkable

performance on a number of tasks and scaling well with increased model and dataset sizes. Such performance is attributed mainly to the multi-head self-attention mechanism in these networks, which allows the network to model interactions between features (called tokens in ViT representation terms) globally at each layer in the network. However, spatial mixing via self-attention is quadratic in the number of tokens, and thus quickly becomes infeasible for high-resolution inputs.

Several ViT variants with reduced computational complexity have been proposed, with various alternate mechanisms for spatial token mixing employed in each. However, the AFNO model is unique in that it frames the mixing operation as continuous global convolution, implemented efficiently in the Fourier domain with FFTs, which allows modeling dependencies across spatial and channel dimensions flexibly and scalably. With such a design, the spatial mixing complexity is reduced to $O(N \log N)$, where N is the number of image patches or tokens. This scaling allows the AFNO model to be well-suited to high-resolution data at the current 0.25° resolution considered in this paper as well as potential future work at an even higher resolution. In the original FNO formulation, the operator learning approach showed impressive results solving turbulent Navier-Stokes systems, so incorporating this into a data-driven atmospheric model is a natural choice.

Given the general popularity of convolutional network architectures, and particularly their usage in previous works forecasting ERA5 variables (Rasp and Thuerey, 2021b; Weyn, Durran, and Caruana, 2020), it is worth contrasting our AFNO model with these more conventional architectures. For one, the ability of AFNO to scale well with resolution yields immediate practical benefits – at our 720x1440 resolution, the FourCastNet model memory footprint is about 10GB with a batch size of 1. To contrast this, we can look at the 19-layer ResNet architecture from a prior result on WeatherBench (Rasp and Thuerey, 2021b), which was trained at a very coarse resolution (32×64 pixels). Naively transferring this architecture to our dataset and training at 720×1440 resolution would require 83GB for a batch size of 1. This is prohibitive, and is compounded by the fact that it is somewhat of a lower bound – with order-of-magnitude increases in resolution, a convolution-based network's receptive field would similarly need to grow via the addition of even more layers.

Beyond practical considerations, our preliminary non-exhaustive experiments suggested that convolutional architectures showed poor performance on capturing small scales over many time steps in auto-regressive inference. These observations along with our knowledge of the current state of the art for high-resolution image process-

ing in image de-noising, super-resolution and de-blurring are a strong motivation for our choice of a ViT architecture over a convolutional architecture.

While we refer the reader to the original AFNO paper (Guibas et al., 2022) for more details, we briefly describe the flow of computation in our model here. First, the input variables on the 720×1440 lat-lon grid are projected to a 2D grid ($h \times w$) of patches (with a small patch size $p \times p$, where e.g., $p = 8$), with each patch represented as a d-dimensional token. Then, the sequence of patches are fed, along with a positional encoding, to a series of AFNO layers. Each layer, given an input tensor of patches $X \in \mathbb{R}^{h \times w \times d}$, performs spatial mixing followed by channel mixing. Spatial mixing happens in the Fourier domain as follows:

Step 1. Transform tokens to the Fourier domain with

$$z_{m,n} = [\mathrm{DFT}(X)]_{m,n}, \tag{11.1}$$

where m, n index the patch location and DFT denotes a 2D discrete Fourier transform.

Step 2. Apply token weighting in the Fourier domain, and promote sparsity with a Soft-Thresholding and Shrinkage operation as

$$\tilde{z}_{m,n} = S_\lambda(\mathrm{MLP}(z_{m,n})), \tag{11.2}$$

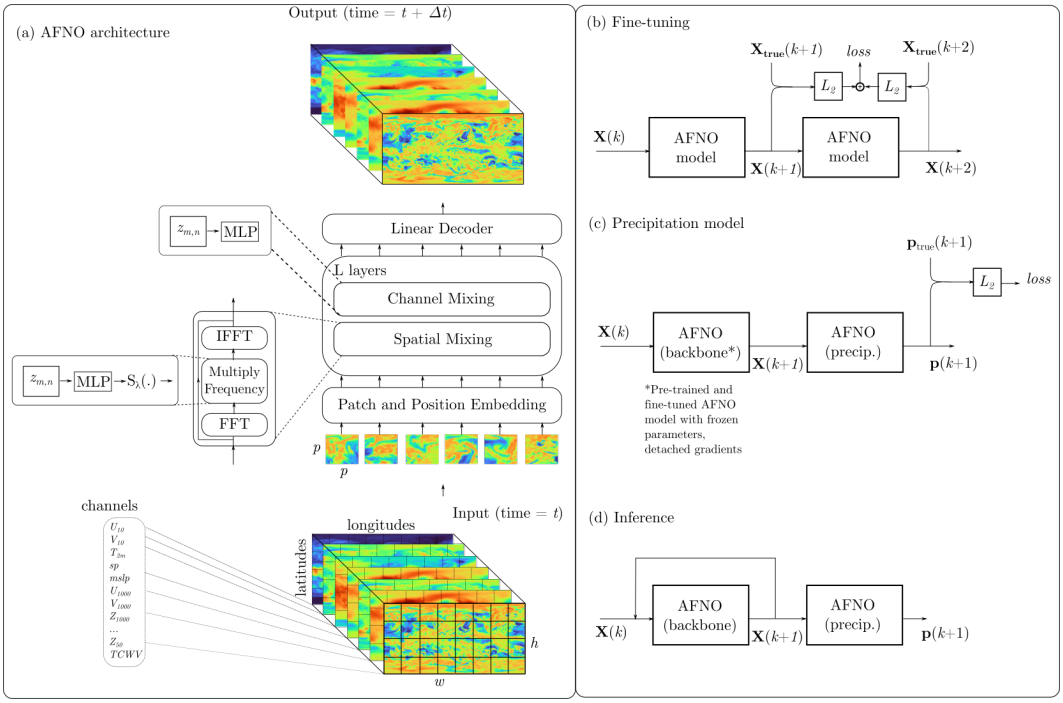
where $S_\lambda(x) = \mathrm{sign}(x) \max(|x| - \lambda, 0)$ with the sparsity controlling parameter λ, and MLP() is a 2-layer multi-layer perceptron with block-diagonal weight matrices which are shared across all patches.

Step 3. Inverse Fourier to transform back to the patch domain and add a residual connection as

$$y_{m,n} = [\mathrm{IDFT}(\tilde{Z})]_{m,n} + X_{m,n}. \tag{11.3}$$

Training

While our primary interest lies in forecasting the surface wind velocities and precipitation, the complex atmospheric system contains strong nonlinear interactions across several variables such as temperatures, surface pressures, humidity, moisture content from the surface of the earth to the stratosphere, etc. In order to model these interactions, we choose a few variables (Table 11.1) to represent the instantaneous



(a) The multi-layer transformer architecture that utilizes the Adaptive Fourier Neural Operator with shared MLP and frequency soft-thresholding for spatial token mixing. The input frame is first divided into a $h \times w$ grid of patches, where each patch has a small size $p \times p \times c$. Each patch is then embedded in a higher dimensional space with high number of latent channels and position embedding is added to form a sequence of tokens. Tokens are then mixed spatially using AFNO, and subsequently for each token the latent channels are mixed. This process is repeated for L layers, and finally a linear decoder reconstructs the patches for the next frame from the final embedding. The right-hand panels describe the FourCastNet model's additional training and inference modes: (b) two-step fine-tuning, (c) backbone model that forecasts the 20 variables in Table 11.1 with secondary precipitation diagnostic model (note that $\mathbf{p}(k+1)$ denotes the 6 hour accumulated total precipitation that falls between $k+1$ and $k+2$ time steps) (d) forecast model in free-running autoregressive inference mode.

Figure 11.2: Architecture of FourCastNet

state of the atmosphere. These variables are specifically chosen to model important processes that influence low-level winds and precipitation. As such, we treat all the prognostic variables equally and the model architecture or optimization scheme does not afford special treatment to any of the prognostic variables.

Each of the variables in Table 11.1 is re-gridded from a Gaussian grid to a regular Euclidean grid using the standard interpolation scheme provided by the Copernicus Climate Data Store (CDS) Application Programming Interface (API). Following the re-gridding process, each of the 20 variables is represented as a 2D field of shape (721×1440) pixels. Thus, a single training data point at an instant in time containing all 20 variables is represented by a tensor of shape ($721 \times 1440 \times 20$). While the ERA5 dataset is available at a temporal resolution of 1 hour, we choose to sub-sample the dataset and use snapshots spaced 6 hours apart to train our model. Within each 24 hour day, we choose to sample the 20 variable subset of the ERA5 dataset at 0000 hrs, 0600 hrs, 1200 hrs and 1800 hrs. We divide the dataset into three sets, namely training, validation and out-of-sample testing datasets. The training dataset consists of data from the year 1979 to 2015 (both included). The validation dataset contains data from the years 2016 and 2017. The out-of-sample testing dataset consists of the years 2018 and beyond.

We collectively denote the modeled variables by the tensor $\vec{X}(k\Delta t)$, where k denotes the time index and Δt is the temporal spacing between consecutive snapshots in the training dataset. We will consider the ERA5 dataset as the truth and denote the *true* variables by $\vec{X}_{\text{true}}(k\Delta t)$. With the understanding that Δt is fixed at 6 hours throughout this work, we omit Δt in our notation for convenience where appropriate. The training procedure consists of two steps, pre-training and fine-tuning. In the pre-training step, we train the AFNO model using the training dataset in a supervised fashion to learn the mapping from $\vec{X}(k)$ to $\vec{X}(k+1)$. In the fine-tuning step, we start from the previously pre-trained model and optimize the model to predict two time steps, i.e., The model first generates the output $\vec{X}(k+1)$ from the input $\vec{X}(k)$. The model then uses its own output $\vec{X}(k+1)$ as an input and generates the output $\vec{X}(k+2)$. We then compute a training loss by comparing each of $\vec{X}(k+1)$ and $\vec{X}(k+2)$ to the respective ground truth from the training data and use the sum of the two training losses for optimizing the model. In both, the pre-training and fine-tuning steps, the training dataset is used to optimize the model and the validation dataset is used to estimate the model skill during hyper-parameter optimization. The out-of-sample testing dataset is untouched. The training dataset consists of 54020 samples while

Vertical Level	Variables
Surface	U_{10}, V_{10}, T_{2m}, sp, $mslp$
$1000hPa$	U, V, Z
$850hPa$	T, U, V, Z, RH
$500hPa$	T, U, V, Z, RH
$50hPa$	Z
Integrated	$TCWV$

Table 11.1: Prognostic variables modeled by the DL model. Abbreviations are as follows. U_{10} (V_{10}): zonal (meridional) wind at 10 m; T_{2m}: temperature at 2 m above ground; T, U, V, Z, RH: temperature, zonal velocity, meridional velocity, geopotential, and relative humidity (at the specified pressure level); $TCWV$: total column water vapor.

the validation dataset contains 2920 samples. We refer to the trained and fine-tuned model as the '*backbone*'. The model is pre-trained using a cosine learning-rate schedule with a starting learning rate ℓ_1 for 80 epochs. Following the pre-training, the model is fine-tuned for a further 50 epochs using a cosine learning-rate schedule and a lower learning rate ℓ_2. The precipitation model (described in Section 11.2) is then added to the trained backbone and trained for 25 epochs using a cosine learning rate schedule with an initial learning rate ℓ_3. The end to end training takes about 16 hours wall-clock time on a cluster of 64 Nvidia A100 GPUs.

Precipitation Model

The total precipitation (TP) in the ERA5 re-analysis dataset is a variable that represents the the accumulated liquid and frozen water that falls to the Earth's surface through rainfall and snow. It is defined in units of length as the depth of water that would accumulate if spread evenly over a unit grid box of the model. Compared to the variables handled by our backbone model, TP exhibits certain features that complicate the task of forecasting it—the probability distribution of TP is strongly peaked at zero with a long tail towards positive values. Hence, TP exhibits more sparse spatial features than the other prognostic variables. In addition, TP does not have significant impact on the variables that guide the dynamical evolution of the atmosphere (e.g. winds, pressures, and temperatures), and capturing it accurately in NWP involves complex parameterizations for processes like phase changes.

For these reasons, we treat the total precipitation (TP) as a diagnostic variable and denote it by $\vec{p}(k\Delta t)$. Total precipitation is not included in the 20 variable

dataset used to train the backbone model[2]. Rather, we train a separate AFNO model to diagnose TP using the outputs of the backbone model, as indicated in Figure 11.2(c). This approach decouples the difficulties of modeling precipitation (which typically deteriorates in accuracy fairly quickly) from the general task of forecasting the atmospheric state. In addition, once trained, our diagnostic TP model could potentially be used in conjunction with other forecast models (either traditional NWP or data-driven forecasts).

The model used to diagnose precipitation from the output of the backbone has the same base AFNO architecture, with an additional 2D convolutional layer (with periodic padding) and a ReLU activation as the last layer, used to enforce non-negative precipitation outputs. Since the backbone model makes predictions in 6-hour increments, we train our diagnostic precipitation model to predict the 6-hourly accumulated total precipitation (rather than the 1 hour precipitation in the raw ERA5 data). This also enables easy comparison with the IFS model, which is archived in 6-hour increments and thus also predicts 6-hourly accumulated precipitation. Following (Rasp, Dueben, et al., 2020), we additionally log-transform the precipitation field: $\tilde{TP} = \log(1 + TP/\epsilon)$, with $\epsilon = 1E-5$. Since total precipitation values are highly sparse, this transformation discourages the network from predicting zeros and ensures a less skewed distribution of values. For any comparisons with the IFS model or ERA5 ground truth, we transform TP back to units of length.

Inference

We generate forecasts of the core atmospheric variables in Table 11.1 and the total precipitation by using our trained models in autoregressive inference mode as shown in Figure 11.2(d). The model is initialized with an initial condition ($\vec{X}_{\text{true}}(j)$) from the year 2018 [3] out-of-sample held out dataset for N_f different initial conditions and allowed to freely run iteratively for τ time-steps to generate forecasts $\{\vec{X}_{\text{pred}}(j+i\Delta t)\}_{i=1}^{\tau}$. The initial conditions $\vec{X}_{\text{true}}(j)$ are spaced apart by D days based on a rough estimate of the temporal de-correlation time for each of the variables being forecast. The value of D and N_f is thus different for each of the forecast variables. We also use the IFS forecasts for the year 2018 from The International Grand Global Ensemble (TIGGE) archive for comparative analysis. The archived IFS forecasts, with initial conditions matching the times of corresponding initial

[2]This approach is similar to previous work (Rasp and Thuerey, 2021b), which trained a separate model for precipitation than for the other atmospheric variables.

[3]The year 2018 was chosen from the out-of-sample dataset due to ready availability of IFS forecasts for that year from the TIGGE archive.

conditions for the FourCastNet model forecast, are used for comparing our model's accuracy to that of the IFS model.

11.3 Results

Figure 11.1 qualitatively shows the forecast skill of our FourCastNet model on forecasting the surface wind speeds over the entire globe at a resolution of 0.25°-lat-long. The wind speeds are computed as the magnitude of the surface wind velocity using the zonal and meridonal components of the wind velocity i.e., $\sqrt{\left(U_{10}^2 + V_{10}^2\right)}$ To prepare this figure, we initialized the FourCastNet model with an initial condition from the out-of-sample test dataset. Starting from this initial condition, the model was allowed to run freely for 16 time-steps in inference mode (Figure 11.2(d)). The calendar time-stamp of the initial condition used to generate this forecast was September 8, 2018 at 00:00 UTC. Figure 11.1(a) shows the wind speed at model initialization. Figure 11.1(b) shows the model forecasts at a lead time of 96 hours (upper-panel) and the corresponding true wind speeds at that time (lower-panel). We note that the FourCastNet model is able to forecast the wind speeds upto 96 hours in advance with remarkable fidelity with correct fine-scale features. Notably, this figure illustrates the forecast of the formation and track of a super-typhoon named Mangkhut that is beginning to form in the initialization frame at roughly $10^{\circ}N$ latitude, $210^{\circ}W$ longitude. The model qualitatively tracks with remarkable fidelity the intensification of the typhoon and its track over a period of 4 days. Also of note are three simultaneous named hurricanes (Florence, Issac and Helene) forming in the Atlantic ocean and approaching the eastern coast of North America during the period of this forecast. The FourCastNet model appears to be able to forecast the formation and track of these phenomena remarkably well. We provide a further discussion of hurricane forecasts with a few quantitative results and case studies in Section 11.3.

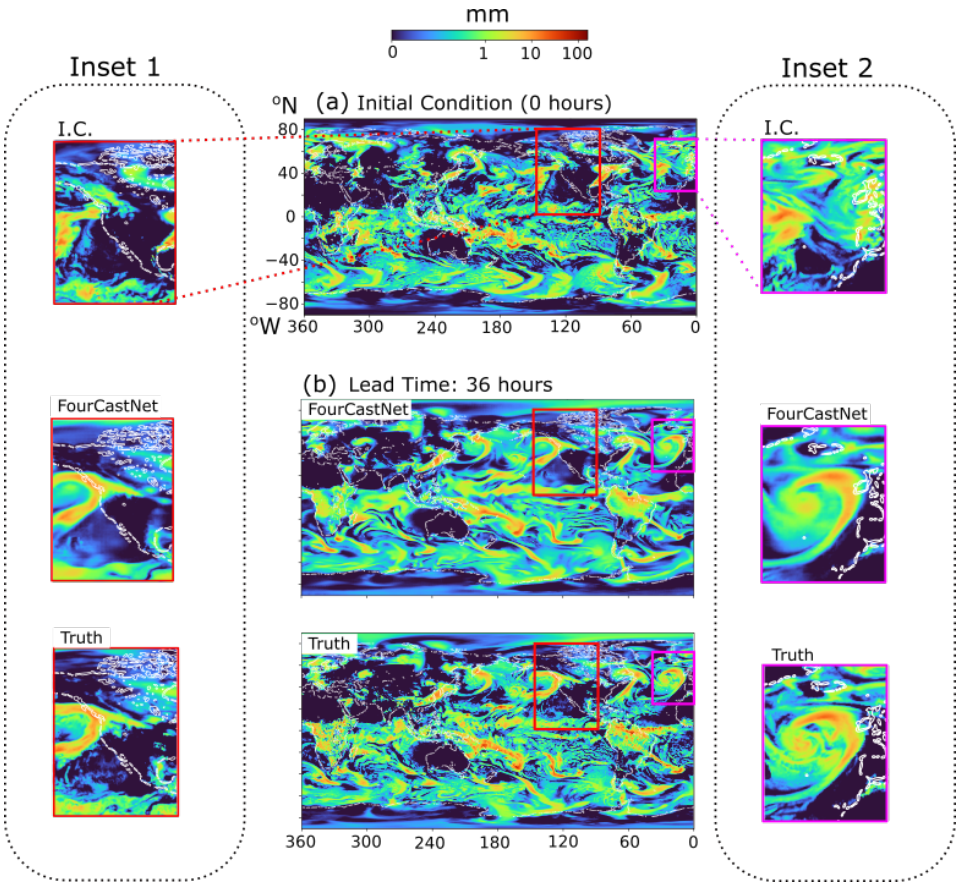
In Fig 11.3, we show the forecast skill of our model in diagnosing total precipitation over the entire globe. Using the free running FourCastNet model predictions (from above) for the 20 prognostic variables as input to the precipitation model, we diagnose total precipitation at the same time steps. Fig 11.3(a) shows the precipitation at the initial time, Fig 11.3(b) shows the model predictions at lead time 36 hours along with the corresponding ground truth. The inset panels show the precipitation fields over a local region along the western coast of the United States, highlighting the ability of the FourCastNet model to resolve and forecast localized areas of high

precipitation with remarkable accuracy. Forecasting precipitation is known to be an extremely difficult task due to its intermittent and stochastic nature. Despite these challenges, we observe that the FourCastNet diagnosis shows excellent skill in capturing short-term high-resolution precipitation features, which can have significant impact in predicting extreme events. We also note that this is the first time a DL model has been successfully utilized to provide competitive precipitation diagnosis at this scale.

Hurricanes

In this section, we explore the potential utility of developing DL models for forecasting hurricanes, a category of extreme events with tremendous destructive potential. A rapidly available, computationally inexpensive atmospheric model that could could forewarn the possibility of hurricane formation and track the path of the hurricane would be of great utility for mitigating loss of life and property damage. As the stakes for mis-forecasting such extreme weather phenomena are very high, more rigorous studies need to be undertaken before DL can be considered a mature technology to forecast hurricanes. The results herein should be considered a preliminary and exploratory dive for inspiring future research into the potential of DL models to provide valuable models of this phenomenon. Prior to this work, DL models were trained on data that was too coarse and thus incapable of resolving atmospheric variables finely enough. Prior models could not generate accurate predictions of wind speed and other important prognostic variables with long enough forecast lead times to consider hurricane forecasts. Our model has reasonably good resolution and generates accurate medium-range forecasts of variables that allow us to track the generation and path of hurricanes. For a case-study we consider a hurricane that occurred in 2018 (a year that is part of our out-of-sample dataset), namely hurricane Michael.

Michael was a category 5 hurricane on the Saffir -Simpson Hurricane Wind Scale that made landfall in Florida causing catastrophic damage (Beven II, Berg, and Hagen, 2019). Michael started as a tropical depression around October 7, 2018. Within a day, the depression intensified into a hurricane. After undergoing rapid intensification in the gulf of Mexico, Michael reached category 5 status. Soon after, Michael made landfall in Florida on October 10, 2018. Thus within a short period of roughly 72 hours, Michael went from a tropical depression to a category 5 hurricane to landfall.



Land-sea borders are shown using a thin white trace. For ease of visualization, the precipitation field is plotted as a log-transformed field in all panels. Panel (a) shows the TP fields at the time of forecast initialization. Panel (b) shows the TP forecast generated by the FourCastNet model (upper panel) over the entire globe at 0.25°-lat-long resolution with the corresponding truth (lower panel). Inset 1 shows the I.C., forecast and true precipitation fields at a lead time of 36 hours over a local region along the western coast of the United States. This highlights the ability of the FourCastNet model to resolve and predict localized regions of high precipitation, in this case due to an atmospheric river. Inset 2 shows the I.C., forecast, and true precipitation fields near the coast of the U.K. and highlights an extreme precipitation event due to an extra-tropical cyclone that is predicted very well by the FourCastNet model. The precipitation is diagnosed from the FourCastNet predicted prognostic variables as described in Figure 11.2(d). The calendar time-stamp of the initial condition used to generate this forecast was 00:00 UTC on April 4, 2018. The high-resolution FourCastNet model demonstrates excellent skill in capturing small scale features that are key to precipitation forecasting.

Figure 11.3: Illustration of a global Total Precipitation (TP) forecast using the FourCastNet model.

We use our trained model as described in Section 11.2 (with no further changes) to study the potential of our model for forecasting the formation, rapid intensification and tracking of hurricane Michael. The FourCastNet model is capable of rapidly generating large ensemble forecasts. We start from the initial condition at the calendar time 00:00 hours on October 7, 2018 UTC. The initial condition was perturbed with Gaussian noise to generate an ensemble of $E = 100$ perturbed initial conditions. We provide further discussion of ensemble forecasting using FourCastNet in Section 11.3. Figure 11.4 shows the track of the hurricane and the intensification as forecast by the 100-member FourCastNet ensemble using the Mean Sea Level Pressure to estimate the eye of the hurricane and the minimum pressure at the eye. Figure 11.4(a) shows the mean position of the minima of Mean Sea Level Pressure using a 100 member ensemble forecast generated by FourCastNet (red circles). The corresponding ground truth according to ERA5 reanalysis is indicated on the same plot (blue squares) over a trajectory spanning 108 hours. The shaded ellipses in the figure have a width and height equal to the 90th percentile spread in the longitudinal and latitudinal positions respectively of the hurricane eye as indicated by the MSLP minima in the 100-member FourCastNet ensemble. Figure 11.4(b) quantitatively demonstrates that the FourCastNet model is able to predict the intensification of the hurricane as the hurricane eye pressure drops rapidly in the first 72 hours. The minimum MSLP at the eye of hurricane Michael as forecast by FourCastNet is indicated by red circles and the corresponding true minimum from the ERA5 reanalysis is shown by blue circles. The red shaded region shows the region between the first and third quartiles of minimum MSLP in the 100-member ensemble. While this is an impressive result for a model trained on 0.25° resolution data, the model fails to fully forecast the extent of the sharp drop in pressure between 36 and 48 hours. We hypothesize that this is likely due to the fact that the current version of the FourCastNet model does not account for a number of convective and radiative processes that would be crucial to such a forecast. Additionally we expect an AFNO model trained on even higher resolution data to improve such a forecast.

Figures 11.4(c),(d) respectively provide a qualitative visualization of three prognostic variables that are useful for tracking the formation, intensification and path of a hurricane, namely the wind speed at the surface and at 850hPa level (calculated as the magnitude of the velocity from the meridional and zonal components of the respective velocity – U_{10}, V_{10}, U_{850}, V_{850}), and the Mean Sea Level Pressure. We believe there is tremendous potential to improve these forecasts by training even

higher resolution DL weather models using the AFNO architecture.

Our forecasts of the wind speeds and the mean sea level pressure qualitatively match the ground truth remarkably well over a period of 72 hours. Figures 11.4(a),(b), (c), (d) clearly show that the DL model is able to forecast the formation, intensification and track of the hurricane from a tropical depression to landfall on the coast of Florida.

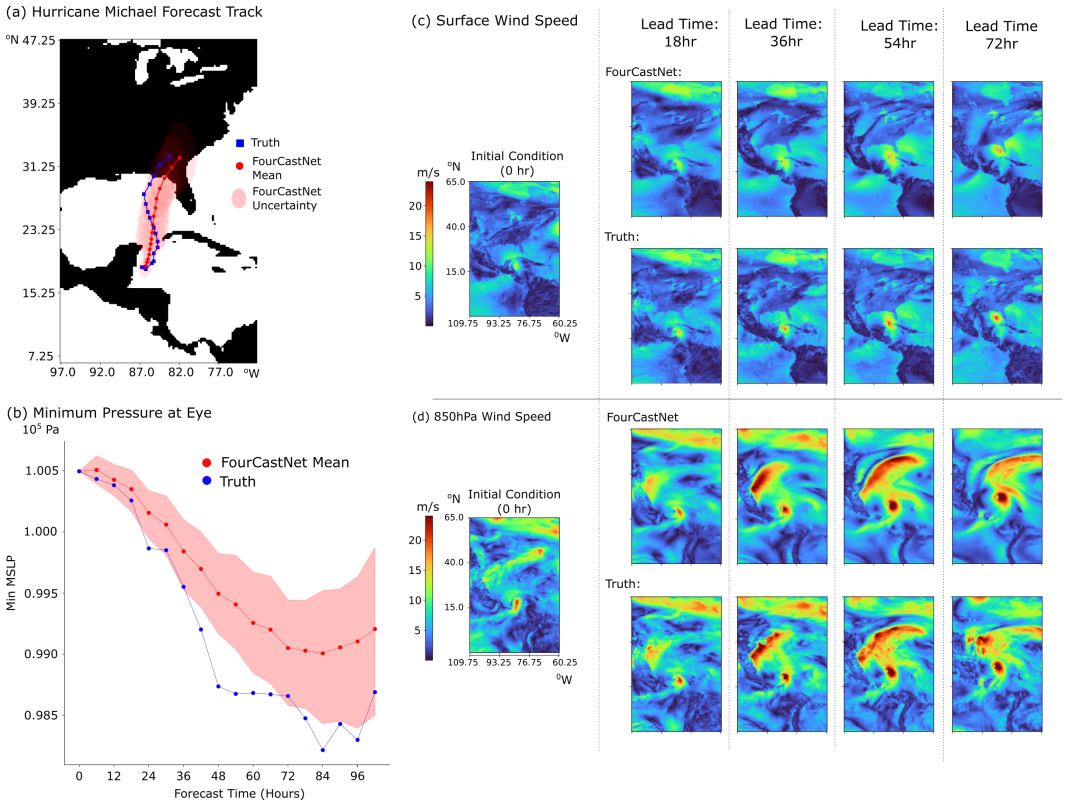
Further research is warranted to quantitatively study the potential of our DL model to accurately forecast hurricanes and similar extreme phenomena but these results show great promise in the ability of DL models to aid in the forecasting of one of the most destructive phenomena affecting human life.

Atmospheric Rivers

Atmospheric rivers are columns of moisture that are transported by atmospheric circulation currents and carry large amounts of water vapor from the tropics to the extra-tropical regions. They are called 'rivers' as they often carry an amount of water equivalent to that of the flow rate of major rivers. Large atmospheric rivers can cause extreme precipitation upon landfall, with the potential to cause flooding and extensive damage. More moderately-sized atmospheric rivers are crucial to the water supply of the western United States. Thus, forecasting atmospheric rivers and their landfall locations is crucial for early warning of flooding in low-lying coastal areas as well as for water resource planning.

Figure 11.5 shows the use of our FourCastNet model for predicting the formation and evolution of an atmospheric river (using the Total Column of Water Vapor variable) in April 2018 as it made eventual landfall in Northern California. This type of river which passes through Hawaii is often called the Pineapple Express. Atmospheric rivers show up very clearly in the 'Total Column Water Vapor' field that is forecast by the FourCastNet backbone model. The FourCastNet model has very good prediction accuracy for $TCWV$, with ACC$>$ 0.6 out beyond 8 days. For this atmospheric river, the FourCastNet model was initialized using an intial condition on April 4, 2018 at 00:00 hours UTC, which we display in Figure 11.5(a). Figures 11.5(b) and 11.5(c) show the forecast of the $TCWV$ fields generated by the FourCastNet model (top panels) at a lead time of 36 hours and 72 hours respectively and the corresponding ground truth (bottom panels).

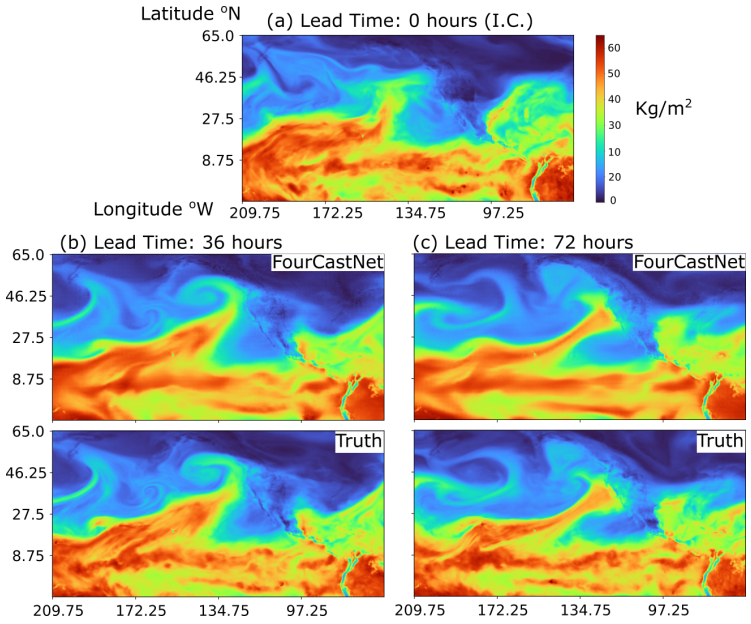
Whie TCWV is a reasonable proxy for atmospheric rivers, we expect future iterations of our model to include Integrated Vapor Transport and Total Column of Liquid



As an illustrative example, we have chosen Hurricane Michael which underwent rapid intensification during the course of its four day trajectory. Panel (a) shows the mean position of the minima of Mean Sea Level Pressure (indicating the eye of hurricane Michael) as forecast by a 100 member ensemble forecast using FourCastNet (red circles) and the corresponding ground truth according to ERA5 reanalysis (blue squares) for 108 hours starting from the initial condition at 00:00 hours on October 7, 2018 UTC. To generate an ensemble forecast, the initial condition was perturbed with Gaussian noise as described in Section 11.3 and 100 forecast trajectories were computed. The shaded ellipses have a width and height equal to the 90th percentile spread of the longitudinal and latitudinal positions respectively of the hurricane eye as indicated by the MSLP minima in the 100-member FourCastNet ensemble. Panel (b) shows the minimum MSLP at the eye of hurricane Michael as forecast by FourCastNet (red filled circles) along with the corresponding true minimum from the ERA5 reanalysis (blue filled circles). The red shaded region shows the 90 percent confidence region in the 100-member ensemble forecast. Panels (c) and (d) respectively show the surface wind speed and 850hPa wind speed predictions at lead times of 18 hours, 36 hours, 54 hours and 72 hours generated by FourCastNet along with the corresponding true wind speeds at those times. The surface wind speed and the 850hPa speed in the initial condition (Oct. 7, 2018 00:00 UTC) that was used to initialize this forecast is shown in the leftmost column. Collectively, the minimum MSLP tracks, surface wind speed and the 850hPa wind speed forecasts show the formation, intensification and path of Hurricane Michael as it goes from a tropical depression to a category 5 hurricane with landfall on the west coast of Florida.

Figure 11.4: The FourCastNet model has excellent skill on forecasting fine-scale, rapidly changing variables relevant to a hurricane forecast.

Water as additional variables to aid in the forecast of atmospheric rivers.



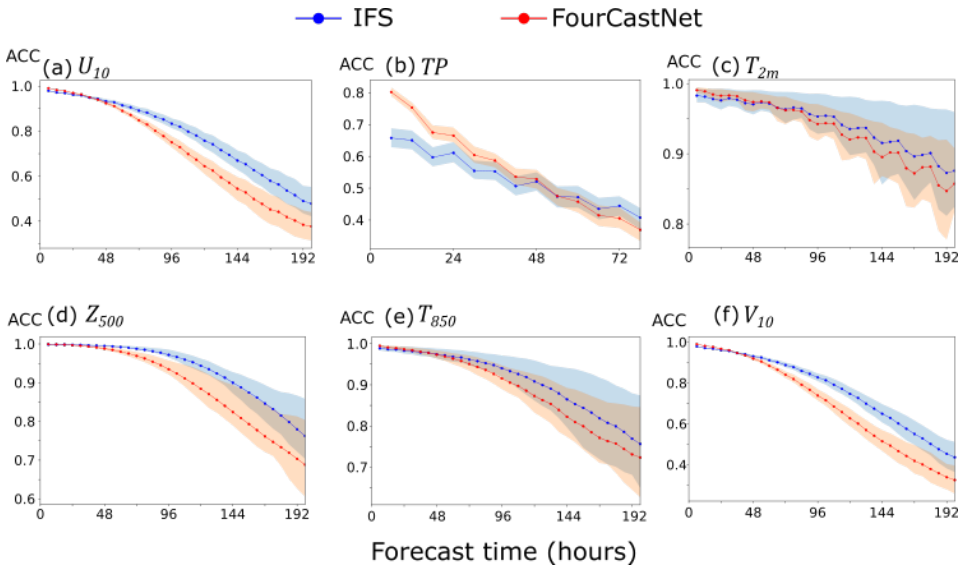
Atmospheric rivers are important phenomena that can cause extreme precipitation and contribute significantly to the supply of precipitable water in several parts of the world. Panels (a)-(c) visualize the Total Column Water Vapor ($TCWV$) in a FourCastNet model forecast initialized at 00:00 UTC on April 4, 2018. Panel (a) Shows the $TCWV$ field in the initial condition that was used to initialize the FourCastNet model. Panels (b) and (c) show the forecasts of the $TCWV$ field produced by the FourCastNet model (top panels) at lead times of 36 and 72 hours respectively along with the corresponding true $TCWV$ fields at those instants of time. The forecast shows an atmospheric river building up and making landfall on the northern California coastline.

Figure 11.5: Illustrative example of the utility of the FourCastNet model for forecasting atmospheric rivers.

Quantitative Skill of FourCastNet

We illustrate the forecast skill of our model for N_f initial conditions from the out-of-sample dataset (consisting of the year 2018) and generate a forecast for each initial condition. For each forecast, we evaluate the latitude-weighted Anomaly Correlation Coefficient (ACC) and Root Mean Squared Error (RMSE) for all of the variables included in the forecast. We report the mean ACC and RMSE for each of the variables along with the first and third quartile values of the ACC and RMSE at each forecast time step, to show the dispersion of these metrics over different initial conditions. As a comparison, we also compute the same ACC and RMSE metrics for the corresponding IFS forecast with time-matched initial conditions.

Figure 11.6(a-f) shows the latitude weighted ACC for the FourCastNet model fore-



averaged over several forecasts initialized using initial conditions in the out-of-sample testing dataset corresponding to the calendar year 2018 for the variables (a) U_{10}, (b) TP, (c) T_{2m}, (d) Z_{500}, (e) T_{850}, and (f) V_{10}. The ACC values are averaged over N_f initial conditions over a full year with an interval of D days between consecutive initial conditions to account for seasonal variability in forecast skill. The appropriately colored shaded regions around the ACC curves indicate the region between the first and third quartile values of the corresponding quantity at each time step.

Figure 11.6: Latitude weighted ACC for the FourCastNet model forecasts (red line with markers) and the corresponding matched IFS forecasts (blue line with markers)

casts (Red line with markers) and the corresponding matched IFS forecasts (Blue line with markers) for the variables (a) U_{10}, (b) TP, (c) T_{2m}, (d) Z_{500}, (e) T_{850}, (f) V_{10}. The ACC and RMSE values are averaged over N_f initial conditions with an interval of D days between consecutive initial conditions. The shaded regions around the ACC curves indicate the region between the first and third quartile values of the corresponding quantity at each time step.

In general, the FourCastNet predictions are very competitive with IFS, with our model achieving similar ACC and RMSE over a horizon of several days. At shorter lead times ($\sim$ 48hrs or less), we actually outperform the IFS model in ACC and/or RMSE for key variables like precipitation, winds, and temperature. Remarkably, we achieve this accuracy using only part of the full variable set available to the IFS model, and we do so at a fraction of the compute cost (see section 11.4 for a detailed speed comparison between models).

Ensemble Forecasts Using FourCastNet

Ensemble forecasts have become a crucial component of numerical weather prediction (Palmer, 2019), and consume the largest share of compute costs at operational weather forecasting centers (Bauer, Quintino, et al., 2020). An ensemble forecast improves upon a single deterministic forecast by modeling multiple possible trajectories of a system. For a chaotic atmosphere with uncertain initial conditions, ensemble forecasting helps quantify the likelihood of extreme events and improves the accuracy of long-term predictions. Thus, rapidly generating large ensemble forecasts is an extremely promising direction for DL-based weather models (Weyn, Durran, Caruana, and Cresswell-Clay, 2021), which can provide immense speedups over traditional NWP models. NWP models such as the IFS perform ensemble forecasting with up to 51 ensemble members. The initial conditions for the ensemble forecasts are obtained by perturbing the analysis state obtained from data assimilation.

As seen in Section 11.3, ensemble forecasting is useful for generating probabilistic forecasts of extreme events such as hurricanes. While the individual perturbed ensemble members typically show lower forecast skill than the unperturbed 'control' forecast, the mean of a large number of such perturbed ensemble members has better forecast skill than the control.

In Section 11.4, we estimate that FourCastNet is roughly 45,000 times faster than a traditional NWP model. This speed allows us to consider probabilistic ensemble forecasting with massive ensemble sizes. Ensemble weather forecasts using FourCastNet are highly computationally efficient because (1.) Inference time for a single forecast on a GPU is very fast and (2.) An ensemble of initial conditions can be folded into the the 'batch' dimension in a tensor and as such, inference on a large batch ($O(100)$ or more) of initial conditions using a few GPUs is straightforward.

As a simple test of ensemble forecasting, we generate an ensemble forecast using FourCastNet from a given ERA5 initial condition by perturbing the initial condition using Gaussian random noise. This allows us to simulate initial condition uncertainty due to errors in the estimate of the starting state of the forecast. This method of ensemble generation is the same as methods used in Ensemble Kalman Filtering (EnKF) (Evensen, 2003) for background forecast covariance estimation and not too dissimilar from the way operational NWP models generate perturbed initial conditions. Thus, given an initial condition $\vec{X}_{\text{true}}(k)$ from our out-of-sample testing dataset, we generate an ensemble of E perturbed initial conditions $\{\vec{X}^{(e)}(k) =$

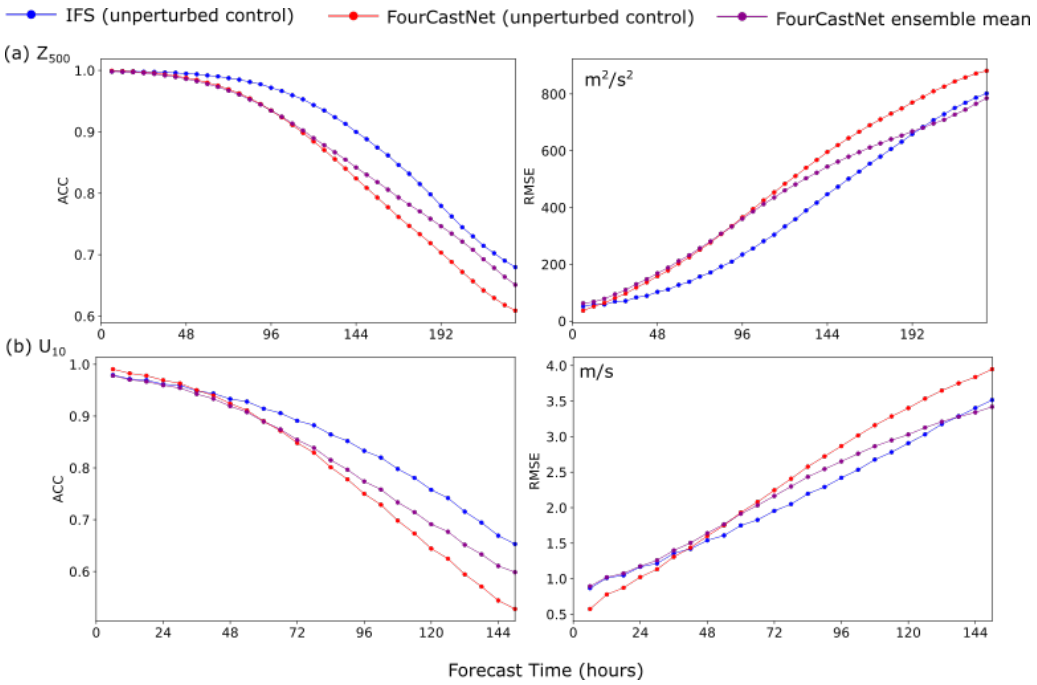
$\hat{\vec{X}}_{\text{true}}(k) + \sigma\xi\}_{e=1}^{E}$, where $\hat{\vec{X}}_{\text{true}}(k)$ is the standardized initial condition with zero mean and unit variance and $\xi \sim \mathcal{N}(\vec{0}, \vec{1})$ is a normally distributed random variable of the same shape as $\vec{X}_{\text{true}}$ and with unit mean and variance. The perturbations are scaled by a factor $\sigma = 0.3$. We refer to the forecast starting from the unperturbed initial condition as the control forecast. We generate an ensemble of perturbed forecasts each starting from a perturbed initial conditions and compute the ensemble mean of the perturbed forecasts at every forecast time step. We compute a control forecast and an ensemble mean forecast for N_f initial conditions separated by D days. We report the mean ACC and RMSE over all N_f initial conditions for both the control and the mean forecast in Figure 11.7.

Figure 11.7 shows the ACC and RMSE of the FourCastNet ensemble mean (magenta line with markers) and FourCastNet unperturbed control (red line with markers) forecasts along with the unperturbed control IFS model (blue line with markers) forecasts for reference. It is challenging to unambiguously visualize in a single plot, both the spread due to simulated initial condition uncertainty in an ensemble forecast and the spread due to seasonal and day-to-day variability. As such, we do not visualize the spread in ACC and RMSE over the N_f forecasts and simply report the mean.

Indeed, in Figure 11.7, we see that the ensemble mean from our 100-member FourCastNet ensemble results in a net improvement in ACC and RMSE at longer timescales over the unperturbed control. We do observe a marginal degradation in skill for the ensemble mean at short ($<$ 48hr) lead times, as averaging over the individual ensemble members likely averages over relevant fine-scale features. Nevertheless, these ensemble forecasts are impressive, and warrant further work in how to optimally choose ensemble members. In addition to perturbing initial conditions with Gaussian noise, as we do here, it is possible and likely worthwhile to introduce more nuanced perturbations to both the initial conditions as well as the model itself. This is a promising direction of research for future work.

Forecast Skill Over Land For Near-surface Wind Speed

Most wind farms are located on land or just off of coastlines, so accurately modeling near-surface wind speed over these regions is of critical importance to wind energy resource planning. To demonstrate the accuracy of FourCastNet predictions over landmasses, we plot the 10m wind speed ($\sqrt{U_{10}^2 + V_{10}^2}$) forecast and ground truth over North America in Figure 11.8. We find that FourCastNet can qualitatively capture

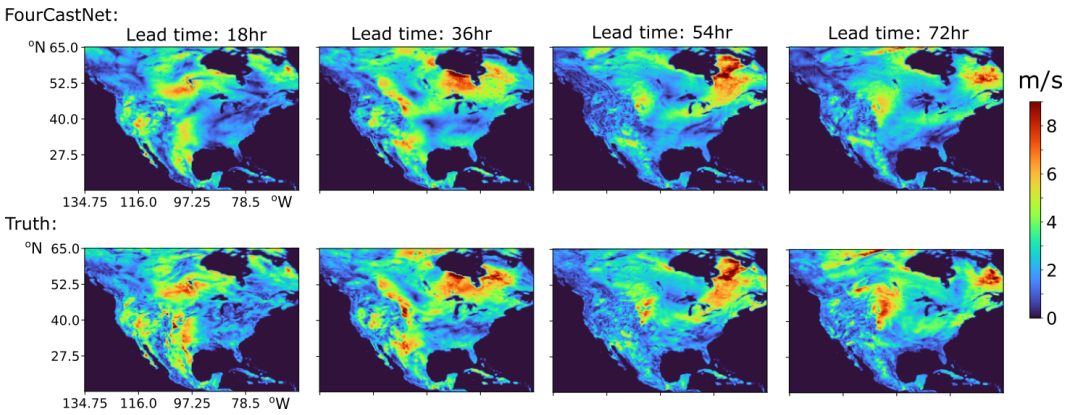


We compare the forecast skill of the unperturbed 'control' forecasts using FourCastNet (red) with the mean of a 100-member ensemble forecast using FourCastNet (magenta) for Z_{500} (panel a) and U_{10} (panel b). The IFS unperturbed control forecast is included for reference (blue). We find that the 100-member FourCastNet ensemble mean is more skillful than the FourCastNet control at longer forecast lead times. The 100-member FourCastNet ensemble mean shows significant improvement over the unperturbed FourCastNet control forecast beyond 70 hours for U_{10} and 100 hours for Z_{500}. Due to the challenge of clearly disambiguating in a single plot the forecast spread arising from simulated initial condition uncertainty and the forecast spread due to seasonal and day-to-day variability, we choose not to visualize the spread in ACC and RMSE over the N_f forecasts and simply report the mean.

Figure 11.7: Illustration of the improvement in forecast skill of FourCastNet by utilizing large ensembles.

the spatial patterns and intensities of surface winds with impressive accuracy up to several days in advance. Moreover, the visualizations emphasize the importance of running forecasts at high resolution, as the surface wind speed exhibits significant fine-scale spatial variations which would be lost with a coarser grid.

We evaluate the forecast skill of our model over land versus over oceans quantitatively. By computing a separate land-masked ACC and a sea-masked ACC for the surface wind velocity components, we find that the forecast quality of our model for surface wind speed over landmass is almost as good as it is over the ocean. This is significant, as surface wind speed over land is strongly affected by orographic features such as mountains, making it in general harder to forecast surface winds



This is a significant result for wind energy resource planning, as windfarms are located on land or just offshore. The figure shows The 10m wind speed ($\sqrt{U_{10}^2 + V_{10}^2}$) forecast (top four panels) generated by FourCastNet and corresponding ground truth (bottom four panels) for forecast lead times of 18 hours, 36 hours, 54 hours and 72 hours. The forecast was initialized with an initial condition at calendar time 06:00:00 on July 4 2018 UTC.

Figure 11.8: The FourCastNet model shows excellent skill on forecasting overland wind speed, a challenging problem due to topographic features such as mountains and lakes.

over land than over the oceans.

Extremes

We assess the ability of the FourCastNet model to capture instantaneous extremes by looking at the top quantiles of each field at a given time step. Similar to the approach in Fildier, W. D. Collins, and Muller, 2021, we use 50 logarithmically-spaced quantile bins $Q = 1 - \{10^{-1}, ..., 10^{-4}\}$ (corresponding to percentiles $\{90\%, ..., 99.99\%\}$) to emphasize the most extreme values (generally, the FourCastNet predictions and ERA5 targets match closely up to around the 98^{th} percentile). We choose the 99.99^{th} as the top percentile bin because percentiles beyond there sample less than 1000 pixels in each image and are subject to more variability. We show example plots of the top quantiles for U_{10} and TP at 24-hour forecast times in the left panel of Figure 11.9 (these particular forecasts were initialized at 00:00 UTC Jan 1 2018). At this particular time, both the FourCastNet and IFS models under-predict extreme precipitation, while for extreme winds in U_{10} the IFS model over-predicts and FourCastNet under-predicts. To get a more comprehensive picture, we need to evaluate the model performance at multiple forecast times over multiple initial conditions in order to ascertain if there is a systematic bias in the model's predictions for extreme values.

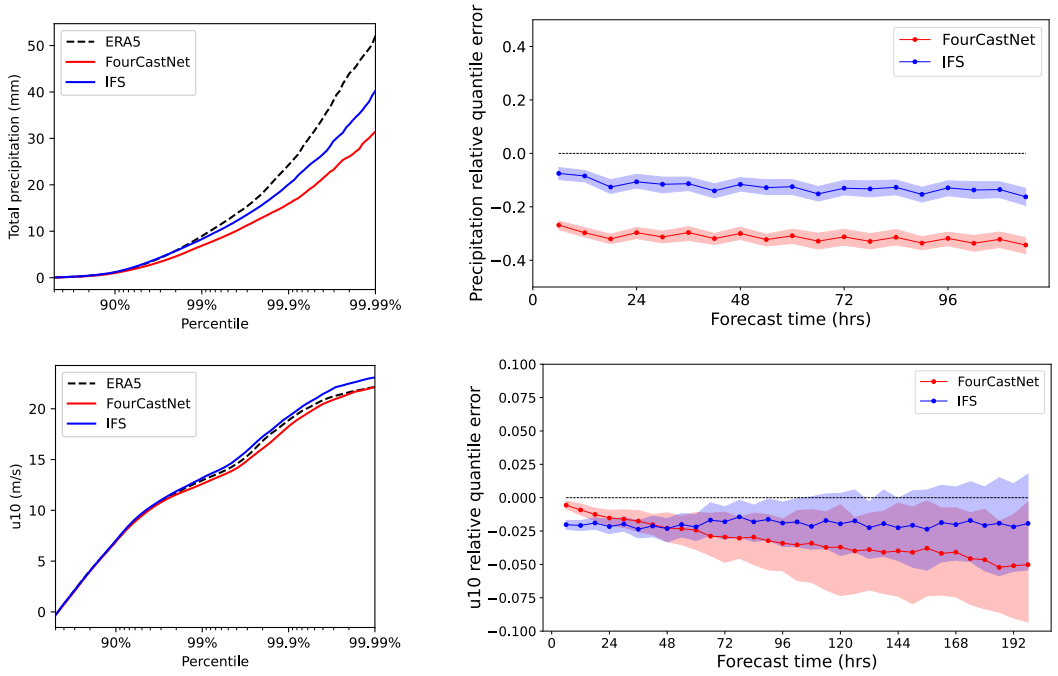


Figure 11.9: Comparison of extreme percentiles between ERA5, FourCastNet, and IFS. The left panel shows the top percentiles of the TP and U_{10} distribution at a forecast time of 24 hours, for a randomly sampled initial condition. The right panel shows the TP and U_{10} relative quantile error (RQE, defined in the text) as a function of forecast time, averaged over N_f initial conditions in the calendar year 2018 (filled region spans the 1st and 3rd quartiles). On average, RQE trends slightly negative for both models as they under-predict the most extreme values for these variables, especially for TP.

To this end, we define the relative quantile error (RQE) at each time step l as

$$\mathrm{RQE}(l) = \sum_{q \in Q} (\vec{X}^{\,q}_{\mathrm{pred}}(l) - \vec{X}^{\,q}_{\mathrm{true}}(l))/\vec{X}^{\,q}_{\mathrm{true}}(l), \tag{11.4}$$

where $\vec{X}^{\,q}(l)$ is the q^{th}-quantile of $\vec{X}(l)$. RQE trends negative for a given variable if a model systematically under-predicts that variable's extremes, and we indeed find that both the FourCastNet and IFS models show a slight negative RQE over different forecast times and initial conditions for both TP and U_{10}. This can be seen in the right-hand panel of Figure 11.9. For U_{10}, the difference between FourCastNet and IFS is negligible and, on average, both models underestimate the extreme percentiles by just a few percentage points in RQE.

For TP, the difference with respect to IFS is more pronounced, and FourCastNet underestimates the extreme percentiles by $\sim 35\%$ in RQE, compared to $\sim 15\%$ for IFS. This is not surprising given the forecasts visualized in Figure 11.3, which show

the FourCastNet predictions being generally smoother than the ERA5 targets. As the extreme values tend to be concentrated in extremely small regions (sometimes down to the gridbox/pixel scale), a model that fails to fully resolve these scales will have a harder time capturing TP extremes. Given the noise and uncertainties, predicting precipitation extremes is well-known to be a challenging problem, but we believe our model could be improved further by focusing more on such fine-scale features. We leave this for future work.

11.4 Computational Cost of FourCastNet

In comparing the speed of forecast generation between FourCastNet and IFS, we have to deal with the rather difficult problem of comparing a forecast computed using a CPU cluster (in the case of the IFS model) and a forecast that is computed on a single (or perhaps a few) GPU(s) (FourCastNet). We take a nuanced approach to reporting this comparison. Our motivation is not to create a definitive apples to apples comparison and tout a single numerical factor advantage for our model, but merely to illustrate the order-of-magnitude differences in forecast generation time and also highlight the radically different perspectives of computation when comparing traditional NWP models with DL models. Through this comparison, we also wish to highlight the significant potential of FourCastNet and future DL models to offer an important addition to the toolkit of a meteorologist.

To estimate the forecast speed of the IFS model, we use figures provided in Bauer, Quintino, et al. (2020) as a baseline. In Ref. (Bauer, Quintino, et al., 2020), we see that the IFS model computes a 15-day, 51-member ensemble forecast using the "L91" 18km resolution grid on 1530 Cray XC40 nodes with dual socket Intel Haswell processors in 82 minutes. The IFS model archived in TIGGE, which we compare the FourCastNet predictions with in Section 11.3, also uses the L91 18km grid for computation (but is archived at the ERA5 resolution of 30km). Based on this information, we estimate that to compute a 24-hour 100-member ensemble forecast, the reference IFS model would require 984,000 node-seconds. We estimate the energy consumption for computing such a 100-member forecast to be 271MJ[4].

We now estimate the latency and energy consumption of the FourCastNet model. The FourCastNet model can compute a 100-member 24-hour forecast in 7 seconds by using a single node on the Perlmutter HPC cluster which contains 4 A100 GPUs per node. This is achieved by performing batched inference on the 4 A100 GPUs

[4] A dual-socket Intel Haswell node draws a Thermal Design Power (TDP) of 270 Watts

using a batch size of 25. Thus, the FourCastNet model takes 7 node-seconds per forecast day for a 100-member ensemble. With a peak power consumption of 1kW per node, we estimate this 100-member 24-hour forecast to use 8kJ of energy.

We attempt to account for the resolution difference between the 18km L91 model and the 30km FourCastNet model by additionally reporting the inference time for an 18km FourCastNet model. Since we did not train the FourCastNet with 18km resolution data (due to the lack of such a publicly available dataset), the reported numbers simply estimate the computational costs for such a hypothetical model by performing inference on data and model parameters interpolated to 18km resolution from the original 30km resolution.

Table 11.2 provides a comparison of computational speed and energy consumption of the IFS L91 18km model and the FourCastNet model at 30km resolution, as well as the extrapolated 18km resolution. These results suggest that the FourCastNet model can compute a 100-member ensemble forecast using vastly fewer nodes, at a speed that is between 45,000 times faster (at the 18 km resolution) and 145,000 times faster (at the 30 km resolution) on a node-to-node comparison. By the same estimates, FourCastNet has an energy consumption that is between 12,000 (18 km) and 24,000 (30 km) times lower than that of the IFS model.

Latency and Energy consumption for a 24-hour 100-member ensemble forecast				
	IFS	FCN - 30km (actual)	FCN - 18km (extrapolated)	IFS / FCN(18km) Ratio
Nodes required	3060	1	2	**1530**
Latency (Node-seconds)	984000	7	22	**44727**
Energy Consumed (kJ)	271000	7	22	**12318**

In comparison, the IFS model needs 3060 nodes for such a forecast. In this table, we provide information about latency and energy consumption for the FourCastNet model in comparison with the IFS model. The FourCastNet model at a 30km resolution is about 145,000 times faster on a single-node basis than the IFS model. We can also estimate the cost of generating an 18km resolution forecast using FourCastNet. Such a hypothetical 18km model would be about 45,000 times faster than the IFS on a single-node basis. The FourCastNet model at 30km resolution uses 24,000 times less energy to compute the ensemble forecast than the IFS model, while a hypothetical FourCastNet model at 18km resolution would use 12000 times less energy.

Table 11.2: The FourCastNet model can compute a 100-member ensemble forecast on a single 4GPU A100 node.

This comparison comes with several caveats. The IFS model generates forecasts that

are provably physically consistent, while FourCastNet in its current iteration does not impose physics constraints. The IFS model also outputs an order of magnitude more variables at as many as 100 vertical levels. Notably, the IFS model is generally more accurate than FourCastNet (although in several variables, the DL model approaches the accuracy of the IFS model and exceeds it for precipitation in certain cases). On the other hand, it is worth noting our rudimentary speed assessments of FourCastNet do not employ any of the common optimizations used for inference of DL models (e.g., model distillation, pruning, quantization, or reduced precision). We expect implementing these would greatly accelerate our speed of inference, and lead to further gains in computational efficiency over IFS.

While the above caveats are important, it is fair to say that if one were only interested in limited-purpose forecasting (e.g., a wind farm operator interested in short-term surface wind speed forecasts), FourCastNet would be a very attractive option as the infrastructure requirements are minimal. FourCastNet can generate a 10-day, global forecast at full ERA5 resolution using a single device, which is simply not possible with IFS, and such a forecast completes in seconds. This means one could generate reasonably accurate forecasts using a tabletop computer with a single GPU, rather than needing a substantial portion of a compute cluster. Similarly, only a handful of GPUs are needed for generating ensemble forecasts with 100s of ensemble members, and such ensembles run quickly and efficiently using batched inference. This greatly lowers the barrier to entry for doing data-assimilation and uncertainty quantification, and future work in this direction is warranted to explore these possibilities.

11.5 Comparison Against State-of-the-art DL Weather Prediction

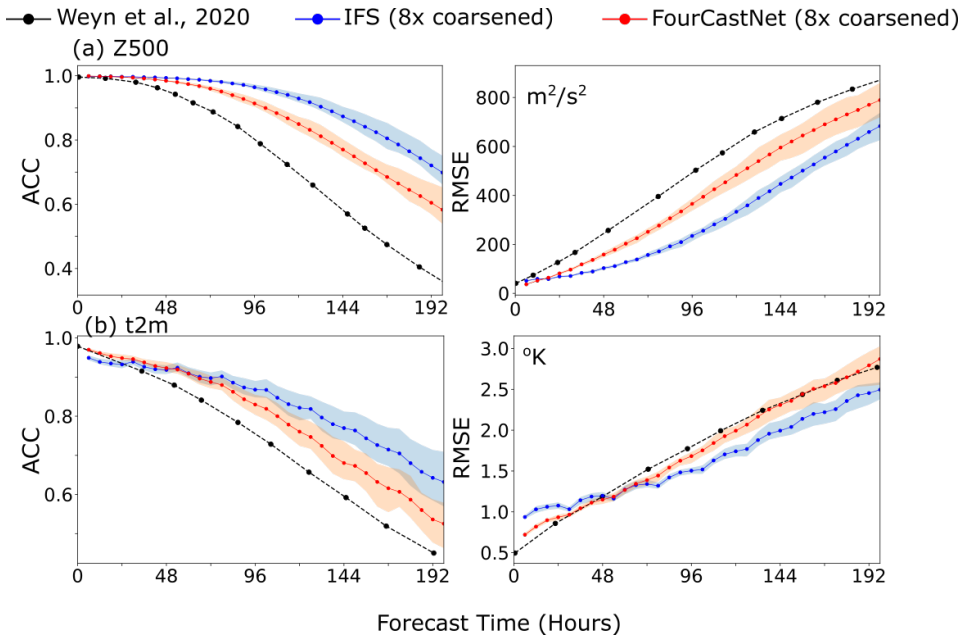
To the best of our knowledge, the current state-of-the-art DL weather prediction model is the DLWP model of Weyn, Durran, and Caruana (2020)—they employ a deep convolutional network with a cubed-sphere remapped coordinate system to predict important weather forecast variables. The authors work with a coarser resolution of 2° and forecast variables relating to geopotential heights, geopotential thickness, and 2-m temperature (see (Weyn, Durran, and Caruana, 2020) for further details). The FourCastNet model predicts more variables than the DLWP model at a resolution that is higher than the DLWP model by a factor of 8. The significantly higher resolution of the FourCastNet model resolves fine-scale features present in variables such as wind velocities and precipitation allowing us to resolve important phenomena such as hurricanes, extreme precipitation and atmospheric rivers. This would not be possible at a lower resolution such as 2° (and almost entirely a futile

exercise at a 5° resolution.) For reference, we have visualized the MSLP over the trajectory of hurricane Michael at a resolution of 2°. Thus, the FourCastNet model has many characteristics that make it superior to the prior SOTA DLWP model. Nonetheless, we undertake a comparison of forecasts generated by the FourCastNet model with those of the DLWP model by coarsening the FourCastNet outputs to bring them to a resolution comparable to that of the DLWP model. We emphasize that this comparison has been provided only for the sake of completeness. Coarsening our forecasts and making them less effective in order to accommodate a prior benchmark at a lower resolution is not fair to our model.

We downsample our predictions eight times (in each direction, using bilinear interpolation) to coarsen them to a resolution that is comparable to that of the DLWP model. Since the two variables reported in the DLWP results are Z_{500} and T_{2m}, we re-compute our ACC and RMSE metrics for those two variables. We also note that the ACC metric in the DLWP baseline was computed using daily climatology (we use a time-averaged climatology in this work, motivated by (Rasp, Dueben, et al., 2020)) and, hence, we modify our ACC computation using the same definition for a fair comparison. We show our comparisons for ACC and RMSE in Figure 11.10. We observe that even at the lower resolution of the DLWP work, the FourCastNet model predictions show significant improvement over the current state-of-the-art DLWP model in both variables. Additionally, the FourCastNet model operates at a resolution that is 8 times higher than the DLWP model allowing it to resolve many important fine-scale phenomena.

11.6 Implications, Discussion, and Future Work

FourCastNet is a novel global data-driven DL-based weather forecasting model based on the FNO and AFNO (Li, Kovachki, et al., 2021; Guibas et al., 2022). FourCastNet's speed, computational cost, energy footprint, and capacity for generating large ensembles has several important implications for science and society. In particular, FourCastNet's high-resolution, high-fidelity wind and precipitation forecasts are of tremendous value. Even though FourCastNet was developed in less than a year and has only a fraction of the number of variables and vertical levels compared to NWP, its accuracy is comparable to the IFS model and better than state-of-the-art DL weather prediction models (Weyn, Durran, Caruana, and Cresswell-Clay, 2021; Rasp, Dueben, et al., 2020) on short timescales. We anticipate that with additional resources and further development, FourCastNet could match the capabilities of current NWP models on all timescales and at all vertical levels of the atmosphere.



(a) Z_{500} and (b) T_{2m}. We observe that the FourCastNet predictions show significant improvement over the baseline model. We also note that the FourCastNet generates predictions that have a higher resolution by a factor of 8, and is thus able to resolve many more important fine-scale features than the DLWP model.

Figure 11.10: Comparison of ACC and RMSE metrics between the (downsampled) FourCastNet predictions, (downsampled) IFS, and baseline state-of-the-art DLWP model

Implications

FourCastNet's predictions are four to five orders of magnitude faster than traditional NWP models. This has two important implications. First, large ensembles of thousands of members can be generated in seconds, thus enabling estimation of well-calibrated and constrained uncertainties in extremes with higher confidence than current NWP ensembles that have at most approximately 50 members owing to their high computational cost. Fast generation of 1,000-member ensembles dramatically changes what is possible in probabilistic weather forecasting, including improving reliability of early warnings of extreme weather events and enabling rapid assessment of their impacts. Second, FourCastNet is suitable for rapidly testing hypotheses about mechanisms of weather variability and their predictability.

The unprecedented accuracy in short-range forecasts of precipitation and its extremes has potentially massive benefits for society such as enabling rapid responses for disaster mitigation. Furthermore, a highly accurate DL-based *diagnostic* precipitation model provides the flexibility to input prognostic variables from different

models or observational sources.

For the wind energy industry, FourCastNet's rapid and reliable high-resolution wind forecasts can help mitigate disasters from extreme wind events and enables planning for fluctuations in wind power output. Wind farm designers can benefit from fast and reliable high-resolution wind forecasts to optimize wind farm layouts that account for a wide variety of wind and weather conditions.

Discussion

FourCastNet's skill improves with increasing number of modeled variables. A larger model trained on more variables, perhaps even on entire 3D atmospheric fields, may extend prediction horizons even further and with better uncertainty estimates. Not far in the future, FourCastNet could be trained on all nine petabytes of the ERA5 dataset to predict all currently predicted variables in NWP at all atmospheric levels. Although the cost of training such a model will be huge, fast inference will enable rapid predictions of entire 3D fields in a few seconds. Such an advancement will likely revolutionize weather prediction.

Due to the current absence of a data-assimilation component, FourCastNet cannot yet generate up-to-the-minute weather forecasts. If observations are available, however, such a component could be readily added given the ease of generating large ensembles for methods such as Ensemble Kalman Filtering with data-driven background covariance estimation (Ashesh Chattopadhyay, Mustafa Mustafa, Hassanzadeh, and Karthik Kashinath, 2020). Therefore, in principle, future iterations of FourCastNet could be trained on observational data. This will enable real-time weather prediction by initializing the model with real-time observations.

With ever-increasing demands for very high-resolution forecasts, NWP has seen a steady growth in resolution. The increase in computational cost of NWP for a doubling of resolution is nearly 12-fold ($2^{3.5}$). Current IFS forecasts are at 9-km resolution but we require forecasts at sub-km resolution for improvements in a wide variety of applications, such as energy and agricultural planning, transportation, and disaster mitigation. Simultaneously, DL continues to have ever-increasing accuracy and predictive power with larger models that have hundreds of billions of parameters (Rajbhandari et al., 2020). With advances in large-scale DL we expect that FourCastNet can be trained to predict weather on sub-km scales. Even though training such a large DL model will be computationally expensive, since inference of large DL models can still be done rapidly (*DeepSpeed: Accelerating large-scale*

model inference and training via system optimizations and compression 2021), a sub-km resolution version of FourCastNet will have even more dramatic speedup over sub-km resolution NWP, likely more than six orders of magnitude.

DLWP has shown good skill on S2S timescales (Weyn, Durran, Caruana, and Cresswell-Clay, 2021). FourCastNet has better skill at short timescales (up to two weeks). We envision a coupled model using a two-timescale approach that combines DLWP and FourCastNet with two-way interactions to achieve unprecedented accuracies on short-, medium-, and long-range weather forecasts.

FourCastNet is a purely data-driven DL weather model. The physical systems of weather and climate are governed by the laws of nature, some of which are well-understood, such as Navier-Stokes equations for the fluid dynamics of atmosphere and oceans. Weather forecasts and climate predictions that obey known physical laws are more trustworthy than those that do not. Furthermore, models that obey the laws of physics are more likely to be robust under climate change. An emerging field in AI applications in the sciences is *Physics-informed Machine Learning* (Kashinath et al., 2021). The Fourier Neural Operator has been extended to be physics-informed (Li, Zheng, et al., 2021). Future versions of FourCastNet will incorporate physical laws. A physics-informed version of FourCastNet could be trained with fewer datapoints. This benefit is particularly valuable at higher resolutions in order to reduce the data volume requirements for training. FourCastNet could also combine with a physics-based NWP model Arcomano, Szunyogh, Wikner, et al. (2021), to generate long-term stable forecasts over S2S timescales.

An important question that remains unanswered is whether FourCastNet generalizes under climate change. FourCastNet was trained on data from 1979 to 2015 and tested on data from 2016 to 2020. We know that Earth's climate has changed over this period of time. Therefore, FourCastNet has been trained on data from a changing climate. However, FourCastNet may not predict weather reliably under extreme climate change expected in the decades to come. A future version will initialize with climate model output to evaluate FourCastNet's performance under different warming scenarios. A grand challenge for the climate community is to predict the changing behavior of extreme weather events under climate change, such as their frequency, intensity, and spatio-temporal nature. Once FourCastNet achieves high fidelity under extreme climate change, it can address this grand challenge.

References

Alley, Richard B, Kerry A Emanuel, and Fuqing Zhang (2019). "Advances in weather prediction". In: *Science* 363.6425, pp. 342–344.

Arcomano, Troy, Istvan Szunyogh, Jaideep Pathak, et al. (2020). "A machine learning-based global atmospheric forecast model". In: *Geophysical Research Letters* 47.9, e2020GL087776.

Arcomano, Troy, Istvan Szunyogh, Alexander Wikner, et al. (2021). "A Hybrid Approach to Atmospheric Modeling that Combines Machine Learning with a Physics-Based Numerical Model". In: *Journal of Advances in Modeling Earth Systems*.

Balaji, V (2021). "Climbing down Charney's ladder: machine learning and the post-Dennard era of computational climate science". In: *Philosophical Transactions of the Royal Society A* 379.2194, p. 20200085.

Bauer, Peter, Tiago Quintino, et al. (2020). *The ecmwf scalability programme: Progress and plans*. European Centre for Medium Range Weather Forecasts. DOI: 10.21957/gdit22ulm. URL: https://www.ecmwf.int/node/19380.

Bauer, Peter, Alan Thorpe, and Gilbert Brunet (2015). "The quiet revolution of numerical weather prediction". In: *Nature* 525.7567, pp. 47–55.

Beven II, J.L., R. Berg, and A. Hagen (Apr. 2019). *Tropical Cyclone Report Hurricane Michael*.

Chantry, Matthew et al. (2021). "Opportunities and challenges for machine learning in weather and climate modelling: hard, medium and soft AI". In: *Philosophical Transactions of the Royal Society A* 379.2194, p. 20200083.

Chattopadhyay, Ashesh, Mustafa Mustafa, Pedram Hassanzadeh, Eviatar Bach, et al. (2021). "Towards physically consistent data-driven weather forecasting: Integrating data assimilation with equivariance-preserving spatial transformers in a case study with ERA5". In: *Geoscientific Model Development Discussions*, pp. 1–23.

Chattopadhyay, Ashesh, Mustafa Mustafa, Pedram Hassanzadeh, and Karthik Kashinath (2020). "Deep spatial transformers for autoregressive data-driven forecasting of geophysical turbulence". In: *Proceedings of the 10th International Conference on Climate Informatics*, pp. 106–112.

Chattopadhyay, Ashesh, Ebrahim Nabizadeh, and Pedram Hassanzadeh (2020). "Analog forecasting of extreme-causing weather patterns using deep learning". In: *Journal of Advances in Modeling Earth Systems* 12.2, e2019MS001958.

DeepSpeed: Accelerating large-scale model inference and training via system optimizations and compression (2021). URL: https://www.microsoft.com/en-us/research/blog/deepspeed-accelerating-large-scale-model-inference-and-training-via-system-optimizatio/ns-and-compression/ (visited on 05/24/2021).

Dosovitskiy, Alexey et al. (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv: 2010.11929 [cs.CV].

Evensen, Geir (2003). "The ensemble Kalman filter: Theoretical formulation and practical implementation". In: *Ocean dynamics* 53.4, pp. 343–367.

Fildier, Benjamin, William D. Collins, and Caroline Muller (2021). "Distortions of the Rain Distribution With Warming, With and Without Self-Aggregation". In: *Journal of Advances in Modeling Earth Systems* 13.2. e2020MS002256 2020MS002256, e2020MS002256. DOI: https://doi.org/10.1029/2020MS002256. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2020MS002256. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2020MS002256.

Grönquist, Peter et al. (2021). "Deep learning for post-processing ensemble weather forecasts". In: *Philosophical Transactions of the Royal Society A* 379.2194, p. 20200092.

Guibas, John et al. (Apr. 2022). "Adaptive Fourier Neural Operators: Efficient Token Mixers for Transformers". In: *International Conference on Representation Learning (to appear)*.

Hersbach, Hans et al. (2020). "The ERA5 global reanalysis". en. In: *Quarterly Journal of the Royal Meteorological Society* 146.730, pp. 1999–2049. ISSN: 1477-870X.

Irrgang, Christopher et al. (2021). "Towards neural Earth system modelling by integrating artificial intelligence in Earth system science". In: *Nature Machine Intelligence* 3.8, pp. 667–674.

Kalnay, Eugenia (2003). *Atmospheric modeling, data assimilation and predictability*. Cambridge University Press.

Kalnay, Eugenia et al. (1996). "The NCEP/NCAR 40-year reanalysis project". In: *Bulletin of the American meteorological Society* 77.3, pp. 437–472.

Kashinath, K et al. (2021). "Physics-informed machine learning: case studies for weather and climate modelling". In: *Philosophical Transactions of the Royal Society A* 379.2194, p. 20200093.

Li, Zongyi, Nikola Kovachki, et al. (2021). "Fourier Neural Operator for Parametric Partial Differential Equations". In: *International Conference on Learning Representations (ICLR)*.

Li, Zongyi, Hongkai Zheng, et al. (2021). *Physics-Informed Neural Operator for Learning Partial Differential Equations*. arXiv: 2111.03794 [cs.LG].

Palmer, Tim (2019). "The ECMWF ensemble prediction system: Looking back (more than) 25 years and projecting forward 25 years". In: *Quarterly Journal of the Royal Meteorological Society* 145.S1, pp. 12–24. DOI: https://doi.org/10.1002/qj.3383. eprint: https://rmets.onlinelibrary.wiley.

com/doi/pdf/10.1002/qj.3383. URL: https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.3383.

Rajbhandari, Samyam et al. (2020). "Zero: Memory optimizations toward training trillion parameter models". In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 1–16.

Rasp, Stephan, Peter D Dueben, et al. (2020). "WeatherBench: a benchmark data set for data-driven weather forecasting". In: *Journal of Advances in Modeling Earth Systems* 12.11, e2020MS002203.

Rasp, Stephan and Nils Thuerey (2020). "Purely data-driven medium-range weather forecasting achieves comparable skill to physical models at similar resolution". In: *arXiv preprint arXiv:2008.08626*.

– (2021a). "Data-Driven Medium-Range Weather Prediction With a Resnet Pretrained on Climate Simulations: A New Model for WeatherBench". In: *Journal of Advances in Modeling Earth Systems* 13.2, e2020MS002405.

– (2021b). "Data-driven medium-range weather prediction with a Resnet pretrained on climate simulations: A new model for WeatherBench". In: *Journal of Advances in Modeling Earth Systems*, e2020MS002405.

Reichstein, Markus et al. (2019). "Deep learning and process understanding for data-driven Earth system science". In: *Nature* 566.7743, pp. 195–204.

Richardson, Lewis Fry (2007). *Weather prediction by numerical process*. Cambridge university press.

Scher, Sebastian and Gabriele Messori (2018). "Predicting weather forecast uncertainty with machine learning". In: *Quarterly Journal of the Royal Meteorological Society* 144.717, pp. 2830–2841.

– (2019). "Weather and climate forecasting with neural networks: using general circulation models (GCMs) with different complexity as a study ground". In: *Geoscientific Model Development* 12.7, pp. 2797–2809.

Schultz, MG et al. (2021). "Can deep learning beat numerical weather prediction?" In: *Philosophical Transactions of the Royal Society A* 379.2194, p. 20200097.

Weyn, Jonathan A, Dale R Durran, and Rich Caruana (2019). "Can machines learn to predict weather? Using deep learning to predict gridded 500-hPa geopotential height from historical weather data". In: *Journal of Advances in Modeling Earth Systems* 11.8, pp. 2680–2693.

– (2020). "Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere". In: *Journal of Advances in Modeling Earth Systems* 12.9, e2020MS002109.

Weyn, Jonathan A, Dale R Durran, Rich Caruana, and Nathaniel Cresswell-Clay (2021). "Sub-seasonal forecasting with a large ensemble of deep-learning weather prediction models". In: *arXiv preprint arXiv:2102.05107*.

Chapter 12

APPLICATION: CARBON CAPTURE AND STORAGE

Carbon capture and storage (CCS) is an essential climate change mitigation strategy for reducing carbon dioxide emissions. We consider the storage aspect of CCS, which involves injecting carbon dioxide into underground reservoirs. This requires accurate and high-resolution predictions of carbon dioxide plume migration and reservoir pressure buildup. However, such modeling is challenging at scale due to the high computational costs of existing numerical methods. We introduce a novel machine learning approach for four-dimensional spatial-temporal modeling, which speeds up predictions nearly 700,000 times compared to existing methods. It provides a general-purpose numerical simulator alternative for highly accurate flow and pressure predictions under diverse reservoir conditions, geological heterogeneity, and injection schemes. Our framework, Nested Fourier Neural Operator (FNO), uses a hierarchy of FNO models to produce outputs at different refinement levels. It enables unprecedented real-time high-resolution modeling for basin-scale carbon dioxide storage.

12.1 Introduction

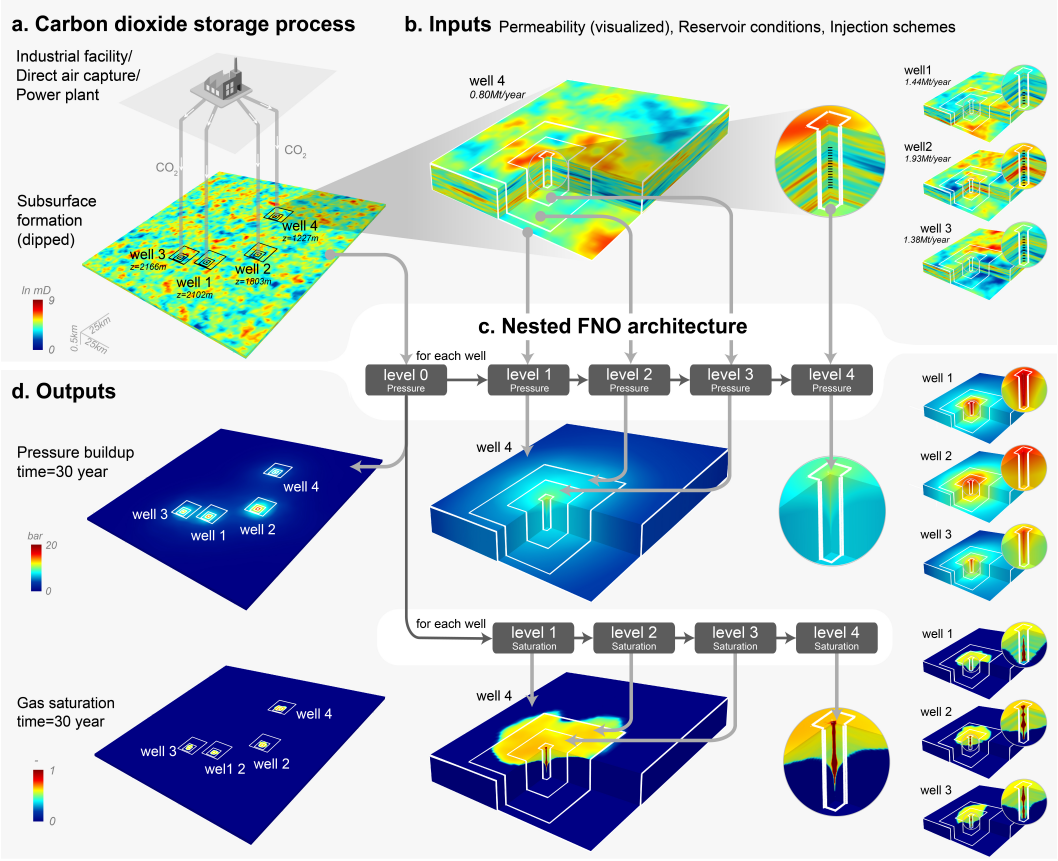
Carbon capture and storage (CCS) is the process of capturing carbon dioxide and permanently storing it in subsurface geological formations to mitigate climate change (NAS, 2018). We consider the storage modeling of CCS, which involves the multiphase flow of carbon dioxide and water through porous media. Numerical simulations are used to guide critical engineering decisions in CCS projects by forecasting carbon dioxide gaseous plume migration in the formation and pressure buildup caused by the injection.

Current numerical simulations are very expensive due to the multi-scale and multi-physics modeling involved, the high-resolution grids required, and the large spatio-temporal ranges needed for real-world scenarios. The governing partial differential equations (PDEs) involving the multiphase variation of Darcy's law are expensive to solve (Pruess, Oldenburg, and Moridis, 1999; Blunt, 2017). Carbon dioxide and water are immiscible and mutually soluble, requiring multi-physics simulation coupled with thermodynamics (Pruess and Garcia, 2002). High-resolution spatial grids (1-2 m resolution) are necessary to provide accurate estimates of the carbon

dioxide plume migration, which is controlled by the complex interplay of buoyancy, viscous, and capillary forces (Doughty, 2010; Wen and Benson, 2019). Pressure buildup, as well as the dry-out effect, i.e., evaporation of formation fluid into the gas phase (Pruess and Müller, 2009; André, Peysson, and Azaroual, 2014), also demand particularly high resolutions around the injection well. In real-world scenarios, CCS projects often involve large reservoir spatial domains of hundreds of kilometers (Chadwick et al., 2004) and long time frames that span from decades to hundreds of years (NETL, 2017).

One approach for reducing the computational costs of numerical simulations is to use non-uniform grids to capture different responses at different grid resolutions. A popular method, known as local grid refinement (Bramble et al., 1988) (LGR), has enabled scaling simulations to real-world three-dimensional (3D) carbon dioxide storage projects, where the fine-grid responses capture the plume migration while the coarser grid responses capture the far-field pressure buildup (Eigestad et al., 2009; Faigle et al., 2014; Kamashev and Amanbek, 2021). However, even with non-uniform grid approaches, these numerical models are still too expensive for tasks that require repetitive forward simulations, e.g., site selection (Callas et al., 2022), inversion (Strandli, Mehnert, and Benson, 2014), or optimization (Nghiem et al., 2010; Zhang and Agarwal, 2012). In practice, these numerical simulation methods are forced to reduce computational costs by coarsening the grid resolution (Kou et al., 2022) and/or simplifying the physics (Cavanagh and Ringrose, 2011), which reduces the accuracy of modeling.

In recent years, machine learning (ML) approaches are emerging as a promising alternative to numerical simulation for subsurface flow problems (Zhu and Zabaras, 2018; Mo et al., 2019; M. Tang, Liu, and Durlofsky, 2020; Wen, M. Tang, and Benson, 2021; Wen, Hay, and Benson, 2021). ML methods are usually much faster than numerical simulators because the inference is very cheap once the ML models are trained. However, standard ML methods suffer from the lack of generalization and fail to provide accurate estimates away from the domain of training data. This limits the usage of ML in challenging applications such as carbon dioxide storage modeling because it requires generalization under diverse geology, reservoir conditions, and injection schemes. A recent machine-learning framework, termed neural operators (Kovachki et al., 2021; Li et al., 2020b; Li et al., 2020c), overcomes these generalization challenges by directly learning the solution operator for the PDE system family instead of just learning a single instance. By learning the solution



(a-b) Permeability for a dipped 3D reservoir with four injection wells; white and black lines indicate level 0 to 4's boundary; the black dotted lines in the zoomed-in circles show the locations of injection perforation intervals. (c) Each grey block represents an FNO model; light grey arrows point to the input and output's level; dark grey arrows show when one model's output is used as another model's input. Notice we feed level 0 pressure buildup output to level 1 gas saturation model because carbon dioxide plumes never migrate to level 0. (d) Pressure buildup and gas saturation at 30 years.

Figure 12.1: Introduction to Nested-FNO.

operator, neural operators generalize well to different conditions in the PDE system as well as discretizations without the need for re-training the ML model.

Fourier neural operator (FNO) is a class of neural operators that uses Fourier transform to learn the solution operator (Li et al., 2020a) efficiently. A variant of FNO was previously used in carbon dioxide storage modeling (Wen, Li, et al., 2022) that showed 60,000 times speedup and excellent generalization. This previous model (Wen, Li, et al., 2022) is limited to a 2D spatial domain that can only represent flat reservoirs with a single injection well. However, In real-world scenarios, CCS projects often involve multiple injection wells and dipped reservoirs.

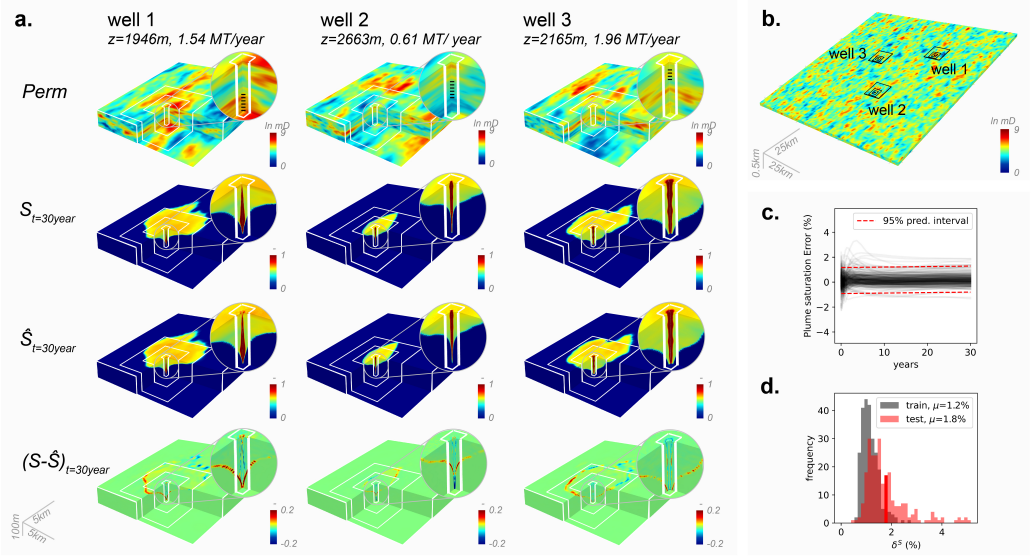
Due to the buoyancy effect, reservoir dipping angles can significantly influence the gaseous plume migration. These processes can only be accurately captured by high-resolution 3D spatial domains, where the costs of collecting training data from numerical solvers become prohibitive. We incorporate these practical modeling features in the current work.

Here, we present an ML approach with unprecedented capabilities of high-resolution 4D spatial-temporal modeling of pressure buildup and gas saturation for realistic carbon dioxide storage projects. We integrate the FNO architecture with the LGR modeling approach to obtain a Nested Fourier Neural Operator (Nested FNO) architecture. As shown in Figure 12.1, five levels of FNOs are used to predict outputs in five levels of grid refinements; each coarser-level model's output is used as the finer-level model's input. This nested approach vastly reduces the computational cost needed during data collection and overcomes the memory constraints in model training. For instance, Nested FNO only needs less than 2,500 training data at the coarsest resolution (level 0) and about 6,000 samples for the finer levels (1-4). Despite the small training size, Nested FNO generalizes well to large problem dimensions with millions of cells. The spatial resolutions provided by Nested FNO are finer than most current simulations run with existing numerical models (e.g., Sleipner benchmark model (*Data: Sleipner CO2 reference dataset, published via the CO2 DataShare online portal administrated by SINTEF AS* n.d.) and Decatur model (*Data: llinois State Geological Survey (ISGS), Illinois Basin - Decatur Project (IBDP) CO2 Injection Monitoring Data, April 30, 2021. Midwest Geological Sequestration Consortium (MGSC) Phase III Data Sets. DOE Cooperative Agreement No. DE-FC26-05NT42588* n.d.)). Nested FNO provides 700,000 times speedup. The fast inference enables rigorous probabilistic assessments for maximum pressure buildup and carbon dioxide plume footprint. Such assessments are important for decisions on injection design and land lease acquisition. Running such assessments take nearly two years with traditional simulators, and it took only 2.8 seconds with the Nested FNO model.

12.2 Methods

Data overview

The data set is generated using a semi-adaptive LGR approach to ensure both high fidelity and computational tractability. We use the global (level 0) resolution grid in the large spatial domain to mimic typical saline storage formations with infinite boundary conditions. Next, we apply four levels of local refinements (levels 1 to



(a) Visualizations of gas saturation predictions at 30 years for a 3-well case. Each row shows permeability, gas saturation ground truth, prediction, and error. The white lines indicate the boundary between each level. (b) Reservoir permeability and the location of each well. (c) Testing set plume saturation error versus time for 250 random cases. The red dotted line shows the 95% prediction bands of the error. (d) Error histograms for 250 cases in the training and test set. The solid red column indicates the error for the shown example.

Figure 12.2: Gas saturation prediction.

4) around each well to gradually increase the grid resolutions. Going from levels 0 to 4, we reduce the cell size by 80x on the x, y dimensions and 10x on the z dimension to resolve near-well plume migration, dry-out, and pressure buildup. See *Supplementary, Local Grid Refinement* for details on the LGR design.

Nested FNO architecture

Nested FNO uses a sequence of FNO models to predict the data set with multiple refinement levels. The computational domain of the Nested FNO is a 3D space with time, $D = \Omega \times T$, where T is the time interval of 30 years and Ω is the reservoir domain. As shown in Figure 12.1, the 3D reservoir domain consists of subdomains Ω_i at levels 0 to 4 for each grid refinement. We use nine FNO models: $\mathcal{G}^P_{0\ldots4}$ for pressure buildup (P), and $\mathcal{G}^S_{1\ldots4}$ for gas saturation (S), to predict outputs at each level. We extend the original FNO (Li et al., 2020a) architecture into 4D to produce outputs in the 3D space-time domain. See *Supplementary, Fourier Neural Operator* for detailed architecture and parameters.

The input for each model includes the permeability field, initial hydro-static pressure,

reservoir temperature, injection scheme, as well as spatial and temporal encoding. In carbon dioxide-water multiphase flow, pressure buildup travels significantly faster than gas saturation. Therefore, we use $\mathcal{G}_0^P$ to predict level 0 (global) pressure buildup as well as the pressure interaction between wells. We then feed level 0 pressure buildup prediction around each injection well ($\hat{P}_0|_{well\,j}$) to models on level 1. Each subsequent model takes the input on domain Ω_i together with the coarser-level prediction of $\hat{P}$ or $\hat{S}$ on Ω_{i-1}, and outputs the prediction of $\hat{P}$ or $\hat{S}$ on Ω_i. By giving the coarser-level prediction to the finer-level model as an input, we also provide the boundary conditions of the finer-level subdomain, which significantly improves the finer-level predictions.

Training procedure

To train the Nested FNO, we first prepare the input-output pairs for each subdomain and train each of the nine models independently. For each model, we use the ground truth, i.e., numerical simulation, pressure buildup and gas saturation on the coarse-level training domain to construct the input. This approach is time efficient because it allows us to train all models concurrently instead of sequentially going from coarser-level to finer-level models. Refer to *Supplementary, Training procedure* for more details.

Inference procedure

Once we train the nine models in the Nested FNO, we can predict the gas saturation and pressure buildup according to Algorithm 5. Notice that the number of subdomains in Ω depends on the number of injection wells. For example, a reservoir with three injection wells has 13 subdomains $\Omega = \{\Omega_0, \Omega_{level1...4,well1}, \Omega_{level1...4,well2}, \Omega_{level1...4,well3}\}$. Therefore, we repeat the inference for each injection well. The input can be constructed given any random combination of reservoir condition (depth, temperature, and dip angle), injection scheme (number of wells, rate, location, perforation interval), and permeability field, as long as the variables are within the training data sampling ranges.

Evaluation metrics

To evaluate the gas saturation prediction accuracy in reservoirs with multiple levels of refinements, we introduce the plume saturation error δ_S, defined as:

Algorithm 5 Predict gas saturation and pressure buildup in a reservoir with n injection wells. $\mathcal{G}$ denotes the a model, P denotes pressure buildup, S denotes gas situation, and a denotes input.

Use $\mathcal{G}_0^P$ to predict $\hat{P}_0$ given a_0
for each well $j = 1, \ldots, n$ **do**
  Construct input $(a_{1,j}, \hat{P}_0|_j)$
  Use $\mathcal{G}_1^P$ and above input to predict $\hat{P}_{1,j}$
  Use $\mathcal{G}_1^S$ and above input to predict $\hat{S}_{1,j}$
  for each level $i = 2, \ldots, 4$ **do**
    Construct input $(a_{i,j}, \hat{S}_{i-1,j})$
    Use $\mathcal{G}_i^S$ and above input to predict $\hat{S}_{i,j}$
    Construct input $(a_{i,j}, \hat{P}_{i-1,j})$
    Use $\mathcal{G}_i^P$ and above input to predict $\hat{P}_{i,j}$
  end for
end for

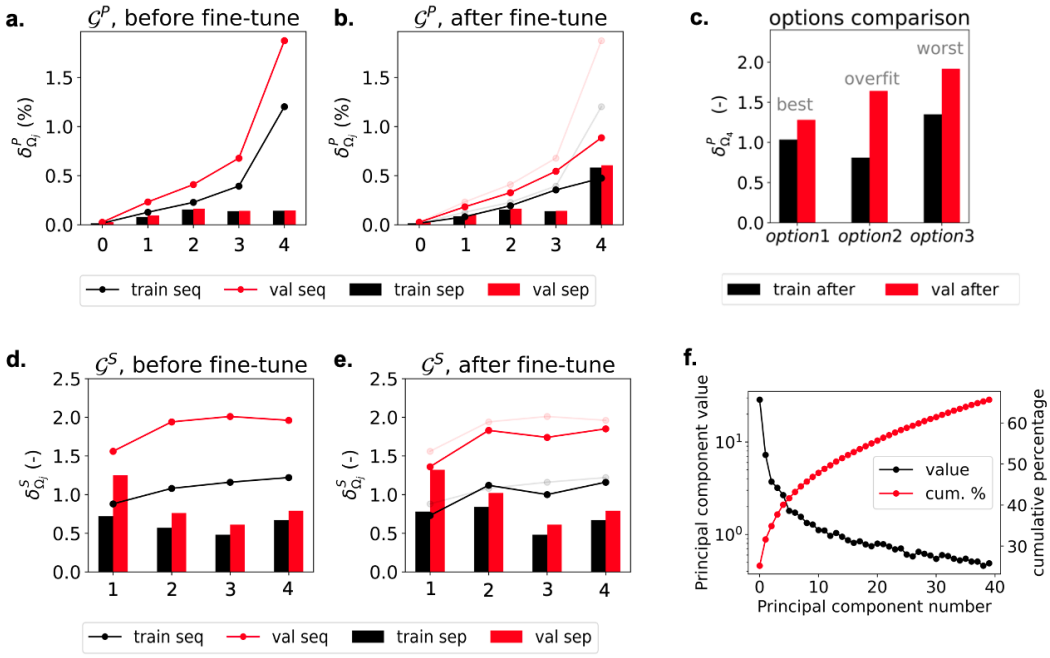
$$\delta^S = \frac{1}{\sum I_{t,i}} \sum_{t \in T} \sum_{i \in \Omega} I_{t,i} |S_{t,i} - \hat{S}_{t,i}|,$$
$$I_{t,i} = 1 \quad if \quad (S_{t,i} > 0.01) \cup (|\hat{S}_{t,i}| > 0.01). \tag{12.1}$$

S is the ground truth gas saturation, $\hat{S}$ is the predicted gas saturation, T includes all times snapshots over the 30 years, and Ω includes all the cells as in the original domain of the numerical simulator; refer *Supplementary, Training procedure* for more details. We use this metric because the reservoir domain includes many cells with zero gas saturation; taking an average with these zero predictions leads to an overestimation of the gas saturation accuracy. δ_S is a more strict metric focusing on the error within the plume.

For pressure buildup, we introduce relative error δ^P:

$$\delta^P = \frac{1}{n_\Omega n_T} \sum_{t \in T} \sum_{i \in \Omega} \frac{|P_{t,i} - \hat{P}_{t,i}|}{P_{t,max}} \tag{12.2}$$

Here P is the ground truth pressure buildup given by numerical simulation, $\hat{P}$ is the predicted pressure buildup, $P_{t,max}$ is the maximum reservoir pressure buildup at time t, n_Ω is the number of cells in Ω, and n_T is the number time steps. This metric is commonly used for evaluating reservoir pressure buildup (H. Tang et al., 2021; Wen, Li, et al., 2022).



Each model's separate and sequential error for (a) pressure buildup before fine-tuning, (b) pressure buildup after fine-tuning, (d) gas saturation before fine-tuning, and (e) gas saturation after fine-tuning. On the legend, 'seq' denotes sequential prediction, 'sep' denotes separate prediction. The transparent lines indicate the before fine-tune error. (e) Training and validation set δ_4^P of fine-tuning using Option 1 to 3. (f) Principle component number and cumulative percentage of the 40 strongest rank for $\mathcal{G}_3^P$'s error.

Figure 12.3: Fine-tuning.

Fine-tuning procedure

Separate vs. sequential prediction. As described in Algorithm 5, during inference, the input for each model in levels 1 to 4 includes the $\hat{S}$ or $\hat{P}$ predicted by their corresponding coarser-level model. However, during training, the inputs are constructed by ground truth numerical simulation data. The discrepancy in training and inference leads to error accumulation, especially for the models that appear later in the prediction sequence.

To investigate this effect, we introduce two ways to evaluate each model: (1) separate prediction using the ground truth input taken from the numerical simulation (as in training), and (2) sequential prediction using predicted values from the coarser level as input (as in inference). Figure 12.3 **a** compares the average relative pressure buildup $\delta^P_{\Omega_j}$ for each model using both separate and sequential prediction methods. Unlike δ^P, $\delta^P_{\Omega_j}$ focuses on the ability of each model to produce outputs similar to the training data, defined as:

$$\delta^P_{\Omega_j} = \frac{1}{n_T n_{\Omega_j}} \sum_{t \in T} \sum_{i \in \Omega_j} \frac{|P_{t,i} - \hat{P}_{t,i}|}{P_{t,max}}. \tag{12.3}$$

Figure 12.3 **a** shows that all models have low errors and negligible overfitting when using separate predictions. However, with sequential prediction, $\delta^P_{\Omega_j}$ quickly accumulates, going from coarser to finer-level models. The validation error of level 4 using sequential prediction increased by 13 times compared to separate predictions.

Similarly, for the gas saturation, the plume gas saturation error $\delta^S_{\Omega_j}$ for each model is defined as:

$$\begin{aligned} \delta^S_{\Omega_j} &= \frac{1}{\sum I_{t,i}} \sum_{t \in T} \sum_{i \in \Omega_j} |S_{t,i} - \hat{S}_{t,i}| \\ I_{t,i} &= 1 \quad if \quad (S_{t,i} > 0.01) \cup (|\hat{S_{t,i}}| > 0.01) \end{aligned} \tag{12.4}$$

Figure 12.3 **d** compares $\delta^S_{\Omega_j}$ using separate verses sequential prediction. We observed less error accumulation for gas saturation than pressure buildup, which indicates that the prediction of gas saturation does not rely as heavily on coarser-level models.

Random perturbation. To reduce the error accumulation, we explored several fine-tuning techniques to improve generalizability using the level 4 pressure prediction as an example. To fine-tune $\mathcal{G}^P_4$, we add a perturbation to the ground truth input, $P'_{3,i} = P_{3,i} + \zeta_i$ where i represents a sample taken from the training set. We defined the coarser-level model's error in the training set as $\epsilon_3 = \hat{P}_3 - P_3$, and explore three configurations of perturbation ζ_i.

- Option 1: $\zeta_i = \epsilon_{3,j}$ - randomly sample an instance from ϵ_3.
- Option 2: $\zeta_i = \epsilon_{3,i}$ - choose the error corresponding to the specific training sample (i.e., fine-tune with the predicted label $\hat{P}_{3,i}$).
- Option 3: $\zeta_i \sim \mathcal{N}(\mu_{\epsilon_3}, \sigma_{\epsilon_3})$ - generate a random Gaussian error using the mean and standard deviation of ϵ_3.

As shown in Figure 12.3 **c**, Option 1 provides the best validation set performance with the smallest overfitting. By providing a randomly sampled noise instance from ϵ_3 with each training data, we let the finer-level models become aware of the

presence of a structured error and learn to filter it out. Option 2 gives the best training set error but is significantly overfitted. Interestingly, Option 3 leads to the largest errors in both the training and validation set despite being a commonly used machine learning technique for introducing randomness.

We applied Option 1 to $\mathcal{G}_1^P$, $\mathcal{G}_4^P$, $\mathcal{G}_1^S$, and $\mathcal{G}_2^S$. After fine-tuning, sequential prediction errors are reduced for both pressure buildup and gas saturation (Figure 12.3 **b** and **e**). For pressure buildup, we observe a dramatic improvement on level 4, where the validation error decreased by more than 50%.

Structure of prediction error. We hypothesize that the FNO model's predictions and errors lie in a perturbed low-dimensional manifold in the output function spaces due to its structure. To verify our hypothesis, we analyzed the functional principle components on error ϵ_3 (Blanchard, Bousquet, and Zwald, 2007). As shown in Figure 12.3 **f**, only a few principal components are needed to describe nearly a third of the error. Gaussian noised functions did not improve prediction because the Gaussian noise resides in an infinite-dimensional space, whereas the actual error only lives in a small linear sub-space.

12.3 Results

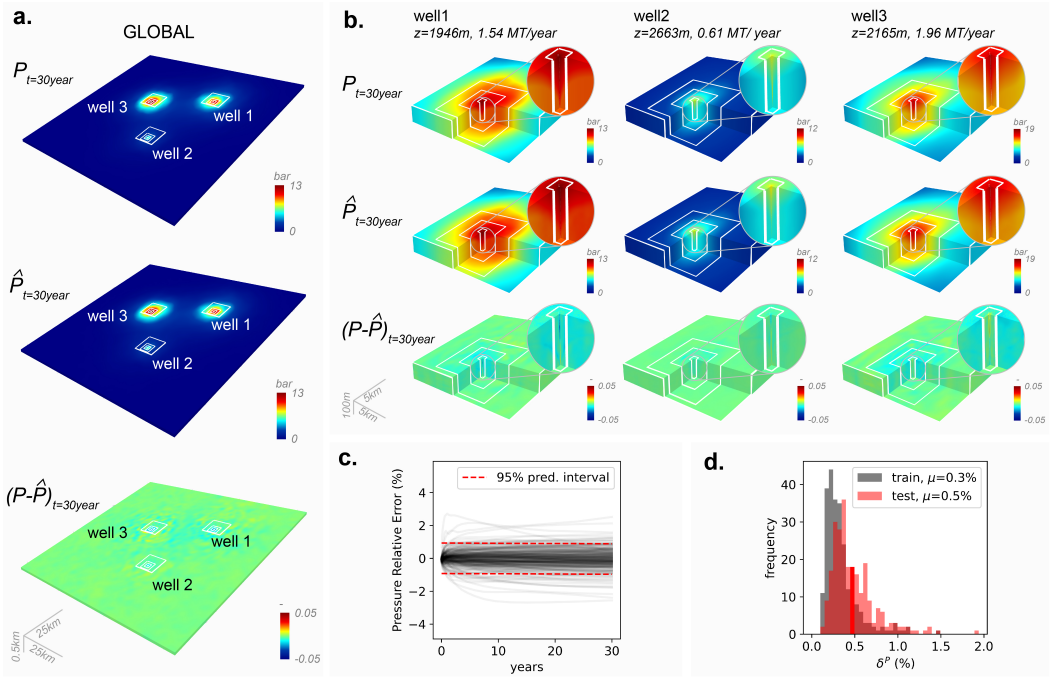
We consider carbon dioxide injection into dipped 3D reservoirs through multiple wells over 30 years, as shown in Figure 12.1 **a**. Our data set considers a comprehensive collection of input variables for practical carbon dioxide storage projects, covering most realistic scenarios of potential sites. Each input variable is sampled according to expert knowledge, including reservoir conditions (depth, temperature, dip angle), injection schemes (number of injection wells, rates, perforation intervals), and permeability heterogeneity (mean, standard deviation, correlation lengths). See *Supplementary, Data set generation* for details for the input variable sampling.

Prediction accuracy

Gas saturation

The migration of carbon dioxide plumes is governed by the complex interplay of viscous, capillary, and gravity forces. Due to the presence of the dip angle, the reservoir condition around each well can lead to various fluid properties in the same reservoir. In addition, carbon dioxide plumes can form distinctively different shapes and sizes according to different injection schemes and permeability heterogeneity.

As shown in Figure 12.2 **a**, Nested FNO captures these complex processes with



(a) Global and (b) well pressure buildup predictions at 30 years. Each row shows pressure buildup ground truth, prediction, and relative error. The white lines indicate the boundary between each level. (c) Testing set pressure relative error versus time for 250 random cases. The red dotted line shows the 95% prediction bands of the error. (d) Error histograms for 250 cases in the training and test set. The solid red column indicates the error for the visualized example.

Figure 12.4: Pressure buildup prediction.

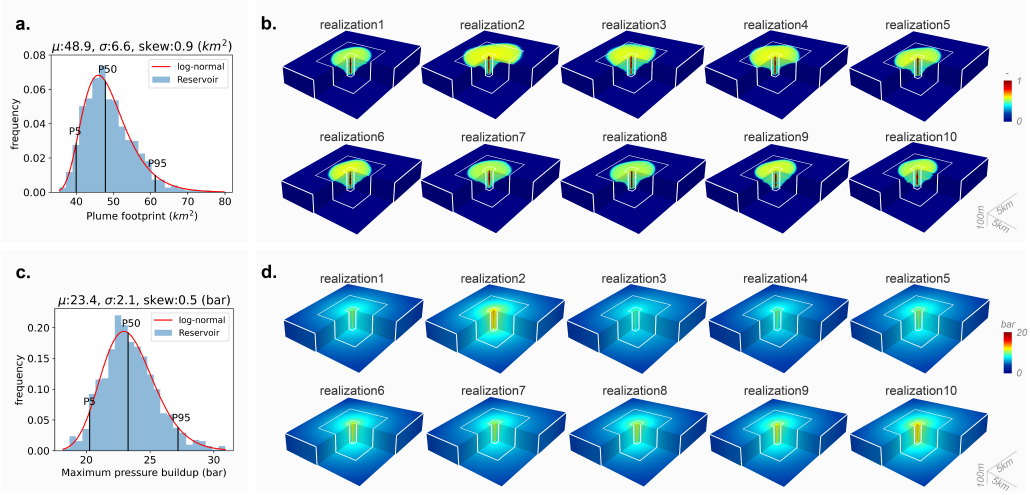
excellent accuracy. Near the injection perforation interval, we observe dry-out zones where the gas saturation is near one, e.g., at the bottom half of well 1. This is a highly nonlinear response caused by the vaporization of the formation fluids into the gas phase (Pruess and Müller, 2009). This important physical process is neglected by many numerical models because it can only be captured with high grid resolutions and high-fidelity simulations. Nevertheless, Nested FNO predicts the dry-out with high accuracy. We observe slightly more errors at the plumes and dry-out zones' edges. This is because the simulation data around discontinuous saturation transitions consist of inherent numerical artifacts that are less systematic.

Overall, Nested FNO displays small overfitting considering the problem's high dimensionality (Figure 12.2 **d**). The average saturation error (δ_S) for the gaseous carbon dioxide plume is 1.2% for the training set and 1.8% for the testing set. See *Methods* for δ_S's definition. This accuracy is sufficient for practical applications such as forecasting plume footprints and estimating sweep efficiencies.

Pressure buildup

As demonstrated in Figure 12.4 **a**, Nested FNO precisely captures the local pressure buildup responses around each well, as well as the global interaction among them. The high resolution provides accurate estimates of the maximum pressure buildup, which is an essential indicator of reservoir integrity. The global level prediction provides the spatial extent of the region of pressure buildup influence, another important parameter required for regulatory purposes (EPA, 2013). Additionally, Figure 12.4 **a-c** shows that accuracy is consistent across different prediction domains and throughout the injection period. These predictions are sufficient to guide important engineering decisions such as injection schemes and monitoring program designs.

The relative pressure buildup error δ^P (as defined in *Methods*) for the training and the testing set are 0.3% and 0.5%, respectively. Similar to the gas saturation, we observe small overfitting from the error histogram (Figure 12.4 **d**) for the training and testing set. This generalization is remarkable, considering the training data size for this high-dimensional problem. The generalizability is achieved through a fine-tuning technique which we will introduce in *Methods*.



(a) Histogram of CO_2 plume footprint predictions given 1,000 permeability realizations from the same geological parameters. The result satisfies a log-normal distribution; P5, P50, and P95 are marked on the distribution. (b) Ten realizations of CO_2 plume at 30 years. (c) Histogram of CO_2 pressure buildup predictions given the same 1,000 permeability realizations. The result satisfies a log-normal distribution; P5, P50, and P95 are marked on the distribution. (d) Ten realizations of pressure buildup at 30 years.

Figure 12.5: Probabilistic assessment.

Computational Speed-up

Once the Nested FNO model is trained, we can use it as a general-purpose numerical simulator alternative in realistic 3D reservoirs. This approach allows traditional users to skip numerical simulations for an entire class of problems and directly obtain efficient, high-fidelity, and high-resolution predictions (Wen, Hay, and Benson, 2021). Our approach differs from the task-specific "surrogate" modeling approach, which often involves a specific set of reservoir conditions.

As a result, we calculate the computational speedup by comparing the Nested FNO's prediction time to the numerical simulation run time of a state-of-the-art full-physics simulator ECLIPSE (Schlumberger, 2014). Each model in Nested FNO takes about 0.005s to infer, while the total prediction time depends on the number of injection wells. On average, the Nested FNO provides 400,000 (1-well case) to 700,000 (4-well case) times speedup compared to traditional simulation. Refer to *Supplementary, Speedup analysis* for detailed specifications for the simulator and ML models.

Probabilistic assessment

Nested FNO's fast prediction speed enables rigorous probabilistic assessments that were previously unattainable. As an example, we investigate the maximum pressure buildup and carbon dioxide plume footprint for a four-well reservoir where each well injects at a 1MT/year rate. Refer to *Supplementary, Probabilistic assessment* for detailed setups. We generate 1,000 permeability realizations using the same distribution and spatial correlations, then use Nested FNO to predict gas saturation plumes and pressure buildup for each realization. As shown in Figure 12.5, the probabilistic estimates of the carbon dioxide plume footprint and maximum pressure buildup can support project developers and regulators in making important engineering decisions (Pawar et al., 2016). For example, the plume footprint helps determine the area of the land lease acquisition required and the monitoring program's design (NETL, 2017); the maximum reservoir pressure buildup helps evaluate the safety of an injection scheme and ensure reservoir integrity. Running such assessments takes nearly two years with traditional numerical simulators, and it takes only 2.8 seconds with Nested FNO.

12.4 Discussion

We present the Nested FNO model for high-resolution 4D gas saturation and pressure buildup in CO_2 storage problems. The trained model provides exceptionally

fast predictions and can support many engineering tasks that require repetitive forward simulations, including but not limited to (1) probabilistic assessment - as demonstrated above, (2) site selection (Callas et al., 2022) - quick screening for a large number of potential reservoirs, (3) storage optimizations (Kumar and Bryant, 2008; Zhang and Agarwal, 2012) - exhaustive search in the parameter space, and (4) seismic inversion (Yin et al., 2022) - provide forward simulation outputs and gradients. Nested FNO can readily serve as a high-fidelity and high-resolution numerical simulator alternative and facilitate rigorous analyses for these tasks.

A highlight of the Nested FNO is its excellent generalizability. The training sizes for Nested FNO are small (2,408 for the level 0 model and 5,916 for level 1-4 models), considering the large problem dimension with millions of cells. We achieve this generalizability through (1) a novel fine-tuning technique for nested architecture as introduced in *Methods* and (2) the utilization of the FNO architecture. Most existing machine learning approaches for subsurface flow use convolutional neural network (CNN)-based models with local kernels and deep architectures that are prone to overfitting (Jiang, Tahmasebi, and Mao, 2021; Wu and Qiao, 2020; Kadeethum et al., 2021; M. Tang, Liu, and Durlofsky, 2021; H. Tang et al., 2021). Unlike CNN, FNO uses global kernels to learn an infinite-dimensional input and output mapping in the function space (Li et al., 2020a). Our previous study shows that FNO-based models require only one-third of the training data compared to the CNN-based model in a 2D spatial-temporal setting (Wen, Li, et al., 2022). As a result, using FNO greatly reduces the demand for training data; combining FNO with the LGR approach makes this high-resolution 3D problem tractable.

Besides data-driven approaches, another line of work, often referred to as a physics-informed neural network, attempts to solve the governing PDE by parameterizing governing relations and initial/boundary conditions using neural networks (Raissi, Perdikaris, and Karniadakis, 2019). However, these approaches have not yet shown significant advantages in computational efficiency for subsurface multiphase flow through heterogeneous media (Fuks and Tchelepi, 2020; Almajid and Abu-Alsaud, 2021; Fraces and Hamdi Tchelepi, 2021; Haghighat, Amini, and Juanes, 2022). On the contrary, as a data-driven approach, Nested FNO provides a framework with great potential not only for CO_2 storage but also for other scientific problems that involve multi-level modeling. For instance, in weather forecast modeling, different cyclones can develop locally while interacting with each other on a global level (Dong and Neumann, 1983); in nuclear fusion, the collision of multiple nuclei

in a particle involves long-distance interactions as well as inner-nucleus many-particle physics (Jin et al., 2021).

References

Almajid, Muhammad M and Moataz O Abu-Alsaud (2021). "Prediction of porous media fluid flow using physics informed neural networks". In: *Journal of Petroleum Science and Engineering*, p. 109205.

André, Laurent, Yannick Peysson, and Mohamed Azaroual (2014). "Well injectivity during CO2 storage operations in deep saline aquifers–Part 2: Numerical simulations of drying, salt deposit mechanisms and role of capillary forces". In: *International Journal of Greenhouse Gas Control* 22, pp. 301–312.

Blanchard, Gilles, Olivier Bousquet, and Laurent Zwald (2007). "Statistical properties of kernel principal component analysis". In: *Machine Learning* 66.2, pp. 259–294.

Blunt, Martin J (2017). *Multiphase flow in permeable media: A pore-scale perspective*. Cambridge university press.

Bramble, James H et al. (1988). "A preconditioning technique for the efficient solution of problems with local grid refinement". In: *Computer Methods in Applied Mechanics and Engineering* 67.2, pp. 149–159.

Callas, Catherine et al. (2022). "Criteria and workflow for selecting depleted hydrocarbon reservoirs for carbon storage". In: *Applied Energy* 324, p. 119668.

Cavanagh, Andrew and Philip Ringrose (2011). "Simulation of CO2 distribution at the In Salah storage site using high-resolution field-scale models". In: *Energy Procedia* 4, pp. 3730–3737.

Chadwick, RA et al. (2004). "Geological reservoir characterization of a CO2 storage site: The Utsira Sand, Sleipner, northern North Sea". In: *Energy* 29.9-10, pp. 1371–1381.

Data: llinois State Geological Survey (ISGS), Illinois Basin - Decatur Project (IBDP) CO2 Injection Monitoring Data, April 30, 2021. Midwest Geological Sequestration Consortium (MGSC) Phase III Data Sets. DOE Cooperative Agreement No. DE-FC26-05NT42588 (n.d.). DOI: 10.11582/2022.00017. URL: https://co2datashare.org/illinois-basin-decatur-project-dataset/static/IBDP%5C%20-%5C%20CO2%5C%20DataShare%5C%20Terms%5C%20Final.pdf.

Data: Sleipner CO2 reference dataset, published via the CO2 DataShare online portal administrated by SINTEF AS (n.d.). DOI: 10.11582/2020.00004. URL: https://co2datashare.org/.

Dong, Keqin and Charles J Neumann (1983). "On the relative motion of binary tropical cyclones". In: *Monthly weather review* 111.5, pp. 945–953.

Doughty, Christine (2010). "Investigation of CO_2 plume behavior for a large-scale pilot test of geologic carbon storage in a saline formation". In: *Transport in porous media* 82.1, pp. 49–76.

Eigestad, Geir Terje et al. (2009). "Geological modeling and simulation of CO2 injection in the Johansen formation". In: *Computational Geosciences* 13.4, pp. 435–450.

EPA (2013). *Geologic Sequestration of Carbon Dioxide - Underground Injection Control (UIC) Program Class VI Well Area of Review Evaluation and Corrective Action Guidance*. 816-R-13-005.

Faigle, Benjamin et al. (2014). "Efficient multiphysics modelling with adaptive grid refinement using a MPFA method". In: *Computational Geosciences* 18.5, pp. 625–636.

Fraces, Cedric G and Hamdi Tchelepi (2021). "Physics Informed Deep Learning for Flow and Transport in Porous Media". In: *SPE Reservoir Simulation Conference*. OnePetro.

Fuks, O and H Tchelepi (2020). "Physics Based Deep Learning for Nonlinear Two-Phase Flow in Porous Media". In: *ECMOR XVII*. Vol. 2020. European Association of Geoscientists & Engineers, pp. 1–10.

Haghighat, Ehsan, Danial Amini, and Ruben Juanes (2022). "Physics-informed neural network simulation of multiphase poroelasticity using stress-split sequential training". In: *Computer Methods in Applied Mechanics and Engineering* 397, p. 115141.

Jiang, Zhihao, Pejman Tahmasebi, and Zhiqiang Mao (2021). "Deep Residual U-net Convolution Neural Networks with Autoregressive Strategy for Fluid Flow Predictions in Large-scale Geosystems". In: *Advances in Water Resources*, p. 103878.

Jin, Shi et al. (2021). "The LISE package: Solvers for static and time-dependent superfluid local density approximation equations in three dimensions". In: *Computer Physics Communications* 269, p. 108130.

Kadeethum, Teeratorn et al. (2021). "A framework for data-driven solution and parameter estimation of PDEs using conditional generative adversarial networks". In: *Nature Computational Science* 1.12, pp. 819–829.

Kamashev, Aibar and Yerlan Amanbek (2021). "Reservoir Simulation of CO2 Storage Using Compositional Flow Model for Geological Formations in Frio Field and Precaspian Basin". In: *Energies* 14.23, p. 8023.

Kou, Zuhao et al. (2022). "Method for upscaling of CO2 migration in 3D heterogeneous geological models". In: *Journal of Hydrology*, p. 128361.

Kovachki, Nikola et al. (2021). "Neural operator: Learning maps between function spaces". In: *arXiv preprint arXiv:2108.08481*.

Kumar, Navanit and Steven Bryant (2008). "Optimizing injection intervals in vertical and horizontal wells for CO2 sequestration". In: *SPE Annual Technical Conference and Exhibition*. OnePetro.

Li, Zongyi et al. (2020a). "Fourier neural operator for parametric partial differential equations". In: *arXiv preprint arXiv:2010.08895*.

– (2020b). "Multipole graph neural operator for parametric partial differential equations". In: *arXiv preprint arXiv:2006.09535*.

– (2020c). "Neural operator: Graph kernel network for partial differential equations". In: *arXiv preprint arXiv:2003.03485*.

Mo, Shaoxing et al. (2019). "Deep convolutional encoder-decoder networks for uncertainty quantification of dynamic multiphase flow in heterogeneous media". In: *Water Resources Research* 55.1, pp. 703–728.

NAS (2018). *Negative Emissions Technologies and Reliable Sequestration*. National Academies Press. ISBN: 9780309484527. DOI: 10.17226/25259.

NETL (2017). *Best Practices: Risk Management and Simulation for Geologic Storage Projects*. URL: https://www.netl.doe.gov/sites/default/files/2018-10/BPM%7B%5C_%7DRiskAnalysisSimulation.pdf.

Nghiem, Long et al. (2010). "Simulation and optimization of trapping processes for CO2 storage in saline aquifers". In: *Journal of Canadian Petroleum Technology* 49.08, pp. 15–22.

Pawar, Rajesh J et al. (2016). "The National Risk Assessment Partnership's integrated assessment model for carbon storage: A tool to support decision making amidst uncertainty". In: *International Journal of Greenhouse Gas Control* 52, pp. 175–189.

Pruess, Karsten and Julio Garcia (2002). "Multiphase flow dynamics during CO2 disposal into saline aquifers". In: *Environmental Geology* 42.2, pp. 282–295.

Pruess, Karsten and Nadja Müller (2009). "Formation dry-out from CO2 injection into saline aquifers: 1. Effects of solids precipitation and their mitigation". In: *Water Resources Research* 45.3.

Pruess, Karsten, Curtis M Oldenburg, and GJ Moridis (1999). *TOUGH2 user's guide version 2*. Tech. rep. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

Raissi, Maziar, Paris Perdikaris, and George E Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378, pp. 686–707.

Schlumberger (2014). *ECLIPSE reservoir simulation software Reference Manual*.

Strandli, Christin W., Edward Mehnert, and Sally M. Benson (2014). "CO_2 plume tracking and history matching using multilevel pressure monitoring at the Illinois basin - Decatur project". In: *Energy Procedia* 63, pp. 4473–4484. ISSN: 18766102. DOI: 10.1016/j.egypro.2014.11.483.

Tang, Hewei et al. (2021). "A deep learning-accelerated data assimilation and forecasting workflow for commercial-scale geologic carbon storage". In: *International Journal of Greenhouse Gas Control* 112, p. 103488.

Tang, Meng, Yimin Liu, and Louis J Durlofsky (2020). "A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems". In: *Journal of Computational Physics* 413, p. 109456.

– (2021). "Deep-learning-based surrogate flow modeling and geological parameterization for data assimilation in 3D subsurface flow". In: *Computer Methods in Applied Mechanics and Engineering* 376, p. 113636.

Wen, Gege and Sally M. Benson (2019). "CO_2 plume migration and dissolution in layered reservoirs". In: *International Journal of Greenhouse Gas Control* 87.May, pp. 66–79. ISSN: 17505836. DOI: 10.1016/j.ijggc.2019.05.012.

Wen, Gege, Catherine Hay, and Sally M. Benson (2021). "CCSNet: a deep learning modeling suite for CO_2 storage". In: *Advances in Water Resources*, p. 104009. ISSN: 0309-1708. DOI: https://doi.org/10.1016/j.advwatres.2021.104009.

Wen, Gege, Zongyi Li, et al. (2022). "U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow". In: *Advances in Water Resources* 163, p. 104180.

Wen, Gege, Meng Tang, and Sally M. Benson (2021). "Towards a predictor for CO_2 plume migration using deep neural networks". In: *International Journal of Greenhouse Gas Control* 105, p. 103223. ISSN: 1750-5836. DOI: 10.1016/j.ijggc.2020.103223.

Wu, Haiyi and Rui Qiao (2020). "Physics-Constrained Deep Learning for Data Assimilation of Subsurface Transport". In: *Energy and AI* 3, p. 100044.

Yin, Ziyi et al. (2022). "Learned coupled inversion for carbon sequestration monitoring and forecasting with Fourier neural operators". In: *arXiv preprint arXiv:2203.14396*.

Zhang, Zheming and Ramesh K Agarwal (2012). "Numerical simulation and optimization of CO2 sequestration in saline aquifers for vertical and horizontal well injection". In: *Computational Geosciences* 16.4, pp. 891–899.

Zhu, Yinhao and Nicholas Zabaras (2018). "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification". In: *Journal of Computational Physics* 366, pp. 415–447.