AI for Scientists: Accelerating Discovery Through Knowledge, Data, and Learning

Thesis by
Jennifer J. Sun

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

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2024
Defended September 15, 2023

ACKNOWLEDGEMENTS

Thanks to everyone.

I am deeply grateful to my co-advisors, Pietro Perona and Yisong Yue. I really appreciate the opportunity. Thank you for your guidance and support.

I would also like to thank my committee members, Ann Kennedy, Swarat Chaudhuri, and Katie Bouman. I have learned lots from working together.

I have also had the luck to meet many wonderful collaborators (across many places) in this process – I hope for many more years of discussions and learning together.

I would also like to thank the members of the Perona Lab and the Yue Crew for their support and encouragement.

I am grateful to the NSERC, Amazon, and the Kortschak Scholars Program for contributing towards the funding for my PhD.

Finally, I would like to thank my family and friends for their love and support throughout this journey. Without them, this thesis would not be possible. Thanks to my parents Debo Sun and Qinghong Li, as well as boyfriend, fiance, husband August Mawn.

ABSTRACT

With rapidly growing amounts of experimental data, machine learning is increasingly crucial for automating scientific data analysis. However, many real-world workflows demand expert-in-the-loop attention and require models that not only interface with data, but also with experts and domain knowledge. My research develops full stack solutions that enable scientists to scalably extract insights from diverse and messy experimental data with minimal supervision. My approaches learn from both data and expert knowledge, while exploiting the right level of domain knowledge for generalization. This thesis presents progress towards developing automated scientist-in-the-loop solutions, including methods that automatically discover meaningful structure from data such as self-supervised keypoints from videos of diverse behaving organisms. We will then discuss methods that use these interpretable structures to inject domain knowledge into the learning process, such as guiding representation learning using symbolic programs of behavioral features computed from keypoints. This work is the result of close collaborations with domain experts, such as behavioral neuroscientists, in order to identify bottlenecks and integrate these methods in real-world workflows. My aim is to enable AI that collaborates with scientists to accelerate the scientific process.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] Jennifer J. Sun, Markus Marks, Andrew Ulmer, Dipam Chakraborty, Brian Geuther, Edward Hayes, Heng Jia, Vivek Kumar, Zachary Partridge, Alice Robie, Catherine E. Schretter, et al. "MABe22: A Multi-Species Multi-Task Benchmark for Learned Representations of Behavior." In: *International Conference on Machine Learning* (2023). URL: `https://arxiv.org/pdf/2207.10553.pdf`.
J.J.S. participated in designing the project, developing the methods, running the experiments, and writing the manuscript.

[2] Jennifer J. Sun*, Lili Karashchuk*, Amil Dravid*, Serim Ryou, Sonia Fereidooni, John C Tuthill, Aggelos Katsaggelos, Bingni W Brunton, Georgia Gkioxari, Ann Kennedy, et al. "BKinD-3D: Self-Supervised 3D Keypoint Discovery from Multi-View Videos." In: (2023), pp. 9001–9010. URL: `https://arxiv.org/pdf/2212.07401.pdf`.
J.J.S. participated in designing the project, developing the methods, running the experiments, and writing the manuscript.

[3] Ting Liu*, Jennifer J. Sun*, Long Zhao, Jiaping Zhao, Liangzhe Yuan, Yuxiao Wang, Liang-Chieh Chen, Florian Schroff, and Hartwig Adam. "View-Invariant, Occlusion-Robust Probabilistic Embedding for Human Pose." In: *International Journal of Computer Vision* 130.1 (2022), pp. 111–135. URL: `https://arxiv.org/pdf/2010.13321.pdf`.
J.J.S. participated in designing the project, developing the methods, running the experiments, and writing the manuscript.

[4] Jennifer J. Sun*, Megan Tjandrasuwita*, Atharva Sehgal*, Armando Solar-Lezama, Swarat Chaudhuri, Yisong Yue, and Omar Costilla-Reyes. "Neurosymbolic Programming for Science." In: *AI for Science Workshop at Neural Information Processing Systems (NeurIPS)* (2022). URL: `https://arxiv.org/pdf/2210.05050.pdf`.
J.J.S. participated in defining the perspective paper, reviewing relevant literature, and writing the manuscript.

[5] Jennifer J. Sun*, Serim Ryou*, Roni H. Goldshmid, Brandon Weissbourd, John O. Dabiri, David J. Anderson, Ann Kennedy, Yisong Yue, and Pietro Perona. "Self-Supervised Keypoint Discovery in Behavioral Videos." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 2171–2180. URL: `https://arxiv.org/pdf/2112.05121.pdf`.
J.J.S. participated in designing the project, developing the methods, running the experiments, and writing the manuscript.

[6] Albert Tseng, Jennifer J. Sun, and Yisong Yue. "Automatic Synthesis of Diverse Weak Supervision Sources for Behavior Analysis." In: *Proceedings*

of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2022), pp. 2211–2220. URL: `https://arxiv.org/pdf/2111.15186.pdf`.
J.J.S. mentored A. T. through this project, and participated in designing the project and methods, and writing the manuscript.

[7] Eric Zhan*, Jennifer J. Sun*, Ann Kennedy, Yisong Yue, and Swarat Chaudhuri. "Unsupervised Learning of Neurosymbolic Encoders." In: *Transactions on Machine Learning Research* (2022). URL: `https://arxiv.org/pdf/2107.13132.pdf`.
J.J.S. participated in designing the project, developing the methods, running the experiments, and writing the manuscript.

[8] Jennifer J. Sun, Ann Kennedy, Eric Zhan, David J. Anderson, Yisong Yue, and Pietro Perona. "Task Programming: Learning Data Efficient Behavior Representations." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 2876–2885. URL: `https://arxiv.org/pdf/2011.13917.pdf`.
J.J.S. participated in designing the project, developing the methods, running the experiments, and writing the manuscript.

[9] Megan Tjandrasuwita, Jennifer J. Sun, Ann Kennedy, Swarat Chaudhuri, and Yisong Yue. "Interpreting Expert Annotation Differences in Animal Behavior." In: *CV4Animals Workshop at the Conference on Computer Vision and Pattern Recognition (CVPR)* (2021). URL: `https://arxiv.org/pdf/2106.06114.pdf`.
J.J.S. mentored M. T. through this project, and participated in designing the project and methods, and writing the manuscript.

[10] Ameesh Shah*, Eric Zhan*, Jennifer J. Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. "Learning Differentiable Programs with Admissible Neural Heuristics." In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 4940–4952. URL: `https://arxiv.org/pdf/2007.12101.pdf`.
J.J.S. participated in project discussions, developing a part of the DSL, running the experiments, and writing the manuscript.

[11] Jennifer J. Sun, Jiaping Zhao, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, and Ting Liu. "View-Invariant Probabilistic Embedding for Human Pose." In: *European Conference on Computer Vision (ECCV)* (2020), pp. 53–70. URL: `https://arxiv.org/pdf/1912.01001.pdf`.
J.J.S. participated in designing the project, developing the methods, running the experiments, and writing the manuscript.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

Introduction

Chapter 1

AI FOR SCIENTISTS

Scientific discoveries have fundamentally changed how we understand the world. These discoveries are built upon observations and experiments, as well as human interpretation to distill data into collective shared insights. The field of ethology is one such example, where observations of moving animals in their environment is distilled into knowledge about the function, evolution, causal, and ontology of their behavior [18]. Ethology has its roots in the work of Konrad Lorenz, Karl von Frisch, and Nikolaas Tinbergen, who were awarded the 1973 Nobel Prize in Physiology / Medicine for their contributions.

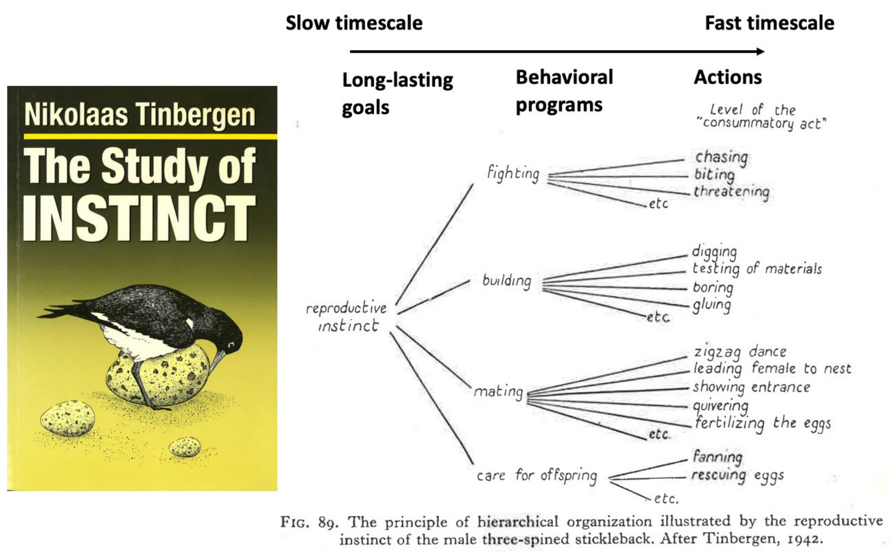


Figure 1.1: Hierarchical behavioral organization of the three-spined stickleback based on [18].

Since these foundational works in the mid-1900s, there has been an increasing movement to quantify animal behavior at greater scales. Traditionally, animal behavior has been translated by human observers into qualitative written descriptions. Following works had human observers who scored incidents of specific behaviors based on expert-defined criteria, using tools such as pencil, paper, and stopwatches; and more recent works have enabled users to score behaviors on computers using keystrokes [1]. Today, the widespread availability of recording equipment and data storage have enabled the creation of large repositories of behavioral data across fields such as pharmacology, neuroscience, and ecology. For example, when neuroscientists study the brain circuitry that regulates animal behavior, one experiment can generate hundreds of hours of videos [14].

While scientific datasets have grown dramatically thanks to technology and automation, human attention has not. Expert attention is still needed to define and categorize actions from these large amounts of videos of behaving animals — 24 hours of recorded videos can take an expert 100 hours to annotate. Manual annotation and analysis is impractical, and automation is increasingly necessary. An automated system that collaborates with scientists to scale their attention to study experimental data at scale have tremendous potential to accelerate the scientific process. We envision a system that could automatically transform large-scale experimental observations into human-readable knowledge as in Figure 1.1, without requiring years of expert time and laborious effort.

Potential of AI in science

In the past few years, rapid developments in machine learning (ML) and computer vision (CV) methods have demonstrated the potential for automating a wide range of tasks previously performed exclusively by humans. Today, these models detect and classify a wide range of objects from image data [19, 6]; enable anyone to generate images and videos from text input [11, 13]; converse with users in natural language, and even could generate new programs [4, 3]. The field has seen significant progress recently, with models now capable of matching or exceeding human performance on many tasks. These new developments have been broadly described as artificial intelligence (AI).

Despite significant progress in AI, many scientific data analysis workflows still rely on human attention – the vision for an AI scientist that automatically learns from data sources around the world to accelerate scientific discovery has not yet been realized. While AI has had significant impacts in fields that rely on computational modeling, such as protein folding and physics simulations, a majority of scientific inquiries do not easily lend themselves to clean mathematical modeling [12, 5, 21], and experts need to work with machines to apply their judgement in order to develop new insights and theory from data and experiments [9, 10, 8].

In this thesis, we will discuss the following questions towards building AI for scientists: (1) What are the biggest challenges from scientific data analysis that motivates new AI development? (2) How do we build AI that's useful to scientists? (3) What are future steps to connect the AI and science research communities? To ground our discussion, let us look at some specific cases in behavior analysis to understand the challenges in analyzing scientific data.

Existing Challenges

Challenge 1: Nonhomogeneous data. Scientific data is highly diverse and is always evolving. For example, in behavioral neuroscience, each lab studies a different subset of behaviors, with different definitions, using different experimental designs. This poses a challenge for our data-hungry models: when a scientist wants to perform a new experiment or study a new behavior, we need to first ask them to annotate tens or even hundreds of thousands of frames of data to train the model. This process is expensive and time-consuming, and the results of the automated system would still need to be manually verified in many cases. As a result, in practice, many labs are still manually annotating all of their data. To address this challenge, we need to develop more data efficient models or models that do not require human annotation to extract useful information.

Challenge 2: Unknown labels. Scientists may not be able to immediately identify what is interesting in new experimental data. Instead, they may need to repeatedly explore the data to find patterns and trends that are significant. This process of iterating over the data is often essential for scientists to discover new insights. For example, when neuroscientists record naturalistic behavior in new experimental settings, it is not always clear which behavioral change might be the result of the experimental intervention or simply due to random variation. We need ML workflows that support rapid iteration and can be adapted to the needs of different scientists — the typical one-shot supervised training process is not always suitable for scientific research.

Challenge 3: Interpretability. Scientists need to be able to trust the results of their models and interpret either the model itself or its outputs. For example, a lab may know exactly which behavior they want to study, and have enough annotated data to train a model, but this is not enough because they also need to be able to explain the distinctive features of each behavior or understand the mechanisms behind a behavior. For example, when studying the gait of a mouse, the lab may be interested in understanding how the animal's gait is affected by different experimental conditions (e.g., pharmacological interventions). In this case, it would not be enough to train a black box model that simply outputs a binary label that the gait is “stable” or “unstable.” Instead, the model needs to be interpretable enough to help scientists understand the differences between gaits. This includes understanding the features that are associated with different gaits, how these features change over time, and how they are affected by different interventions. Through this, scientists can better predict how gait might change in response to new interventions and ultimately connect behavioral changes to biological mechanisms.

Due to these challenges, many existing models and frameworks work well for specific experiments and conditions, but are often difficult to share across labs (even for labs studying the same model organisms). These pipelines are often tailored to the specific data and methods used during development, and need to be re-trained or re-designed in new settings (for example, moving recording from top view to side view camera to capture new behaviors can require retraining multiple models in existing pipelines). Typically, the only way to share the results of these analyses is to publish them in a paper. However, this does not allow other labs to easily reproduce the analyses or to use the models for their own experiments.

To advance the field, we need to develop models that can be shared across labs, are interpretable to scientists in different labs, and generalize to diverse domain-specific data. Now is the time to work on these challenges, as there is increasing pressure from the community and funding agencies for labs to open source their datasets, which has led to the creation of large-scale open source repositories [20]. These repositories contain heterogeneous datasets across labs, which can be used to train and evaluate generalizable models. At the same time, AI development is moving towards foundation models [2] that have the potential to generalize to a variety of downstream tasks with zero/few-shot performance. My work in developing AI for scientists grounds AI development in scientific challenges with real-world stakeholders, as well as help scientists process large-scale heterogeneous datasets to extract insights. My goal is to build towards an AI ecosystem that works across scientific domains to accelerate discovery.

1.1 Themes

How do we build AI that is useful to scientists? This thesis focuses on scientific data analysis[1], where we want to automatically extract insights from large-scale

[1]There are a number of other steps in the scientific process, such as hypothesis design, data collection, or running simulations, which is not the focus of this work.

experimental data. However, there are a number of challenges to working with scientific data as we outlined above.

To tackle these challenges, I aim to develop models that work together with scientists to extract insights from data, by automatically converting data into symbolically interpretable representations. These representations enable scientists to bidirectionally interact with the system, both to inject domain knowledge and to efficiently extract insights. For example, in the case of animal behavior analysis, the raw data consists of videos of animal behavior. The insight that we want to extract is the frame-by-frame label of the animal's behavior. The symbolically interpretable representations could be a set of sparse keypoint locations that track semantically meaningful body parts on the animals. These keypoint locations allow experts to inject domain knowledge into the system (for example, through hand-crafted features such as distance, speed, acceleration), and also be used to correlate with other measurements such as neural recordings.

There are a few themes in my approach:

Grounding algorithm development in real-world workflows. ML models are crucial in scaling analysis in science, and at the same time, scientists pose new and fascinating questions for ML researchers. For example, how to develop models that can be trained and shared across entire communities, where each scientist carries out idiosyncratic experiments?; What is the best way to combine symbolic domain knowledge with data to produce interpretable models and theories? These challenges from real-world settings motivates new algorithms, and prevents overfitting to datasets curated for ML and CV. I approach the algorithm development process by closely collaborating with scientists to understand the bottlenecks and challenges in their workflow, and translating these challenges to computational ones. This process involves understanding the task that experts need to perform, the data to analyze, the current approach, and the resources available for model development and deployment. In my work, I have focused on behavior analysis as a domain, but many of our work applies to other expert-in-the-loop workflows, such as analyzing EKG data for cardiology.

Efficiently use scientists' efforts. In the ideal case, our tools would require minimal effort from scientists to extract maximum insight from data. To use existing tools [7], labs require time and resources to setup computation, annotate data, train new models, and deploy these models. In this workflow, we can reduce expert effort required through more data-efficient models, or models that automatically discovers useful structure from data. Moving forward, AI systems (such as foundation models) that could be deployed to analyze scientific data with zero-shot or few-shot performance has the potential to further reduce expert effort. These systems could potentially interact with experts through natural language to efficiently encode domain knowledge (instead of through time-consuming data annotation).

Leveraging the right domain knowledge to generalize. We would like to build algorithms that work well for specific labs, but at the same time, also generalizes to different experimental setups and tasks. The methods I study in this thesis encode varying levels of domain knowledge to achieve this. Neurosymbolic learning is one type of approach I have explored for encoding domain knowledge, which integrates symbolic knowledge from scientists with the flexibility of neural networks. In this framework, scientists can express prior knowledge into the design of a domain-specific language (DSL), and use neurosymbolic program synthesis to discover these programs. Another way that we explore this theme is by making appropriate assumptions on properties of the experimental data. For example, by observing that almost all behavioral videos are recorded from stationary cameras, we can design a simple self-supervised loss that discovers keypoints from videos of a wide range of organisms [17].

Datasets and Benchmarks. Representative real-world datasets and benchmarks are crucial for model development and for quantitatively evaluating progress. These datasets need to be high fidelity for scientists, but also scalable for evaluation during model development. This thesis will present progress on the first set of datasets and benchmarks in ML for behavioral neuroscience [15, 16], towards a systematic way to measure progress in behavior analysis. Notably, one limitation of standardized benchmarking is that the community may overfit on certain tasks; on the other hand, fragmenting the community across different datasets makes methods difficult to compare quantatitively. One approach I take is to evaluate new methods both on datasets central to the ML & CV community to compare with existing methods, as well as on lab-specific data to measure performance on real-world tasks. Moving forward, maintaining a live and versioned ML for behavior benchmark could enable to community to contribute lab-specific data, and also enable evaluation on more standardized splits.

References

[1] David J. Anderson and Pietro Perona. "Toward a Science of Computational Ethology." In: *Neuron* 84.1 (2014), pp. 18–31.

[2] Rishi Bommasani et al. "On the Opportunities and Risks of Foundation Models." In: *arXiv preprint arXiv:2108.07258* (2021).

[3] Xi Chen et al. "Pali: A Jointly-Scaled Multilingual Language-Image Model." In: *arXiv preprint arXiv:2209.06794* (2022).

[4] Jacob Devlin et al. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding." In: *arXiv preprint arXiv:1810.04805* (2018).

[5] Jared Diamond. "Soft Sciences Are Often Harder Than Hard Sciences." In: *Discover* 8.8 (1987), pp. 34–39.

[6] Kaiming He et al. "Mask R-CNN." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2961–2969.

[7] Kevin Luxem*, Jennifer J. Sun*, Sean P. Bradley, Keerthi Krishnan, Talmo D. Pereira, Eric A. Yttri, Jan Zimmermann, and Mark Laubach. "Open-Source Tools for Behavioral Video Analysis: Setup, Methods, and Development." In: *eLife* (2023).

[8] Michael Muthukrishna and Joseph Henrich. "A Problem in Theory." In: *Nature Human Behaviour* 3.3 (2019), pp. 221–229.

[9] Paul Nurse et al. "Biology Must Generate Ideas as Well as Data." In: *Nature* 597.7876 (2021), pp. 305–305.

[10] Clare Press, Daniel Yon, and Cecilia Heyes. "Building Better Theories." In: *Current Biology* 32.1 (2022), R13–R17.

[11] Aditya Ramesh et al. "Hierarchical Text-Conditional Image Generation with CLIP Latents." In: *arXiv preprint arXiv:2204.06125* 1.2 (2022), p. 3.

[12] Michael C. Reed. "Why is Mathematical Biology so Hard." In: *Notices of the AMS* 51.3 (2004), pp. 338–342.

[13] Chitwan Saharia et al. "Photorealistic Text-To-Image Diffusion Models with Deep Language Understanding." In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 36479–36494.

[14] Cristina Segalin, Jalani Williams, Tomomi Karigo, May Hui, Moriel Zelikowsky, Jennifer J. Sun, Pietro Perona, David J. Anderson, and Ann Kennedy. "The Mouse Action Recognition System (MARS) Software Pipeline for Automated Analysis of Social Behaviors in Mice." In: *eLife* 10 (2021), e63720.

[15] Jennifer J. Sun, Markus Marks, Andrew Ulmer, Dipam Chakraborty, Brian Geuther, Edward Hayes, Heng Jia, Vivek Kumar, Zachary Partridge, Alice Robie, Catherine E. Schretter, et al. "MABe22: A Multi-Species Multi-Task Benchmark for Learned Representations of Behavior." In: *International Conference on Machine Learning* (2023). URL: `https://arxiv.org/pdf/2207.10553.pdf`.

[16] Jennifer J. Sun, Tomomi Karigo, Dipam Chakraborty, Sharada Mohanty, Benjamin Wild, Quan Sun, Chen Chen, David Anderson, Pietro Perona, et al. "The Multi-Agent Behavior Dataset: Mouse Dyadic Social Interactions." In: *Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track* (2021).

[17] Jennifer J. Sun*, Serim Ryou*, Roni H. Goldshmid, Brandon Weissbourd, John O. Dabiri, David J. Anderson, Ann Kennedy, Yisong Yue, and Pietro Perona. "Self-Supervised Keypoint Discovery in Behavioral Videos." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 2171–2180. URL: `https://arxiv.org/pdf/2112.05121.pdf`.

[18] Nikolaas Tinbergen. *The Study of Instinct*. Clarendon Press/Oxford University Press, 1951.

[19] Grant Van Horn et al. "The iNaturalist Species Classification and Detection Dataset." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8769–8778.

[20] Hollie White et al. "The Dryad Data Repository: A Singapore Framework Metadata Architecture in a DSpace Environment." In: *Dublin Core Conference*. 2008, pp. 157–162.

[21] Eric F. Wolstenholme. "Qualitative vs Quantitative Modelling: The Evolving Balance." In: *Journal of the Operational Research Society* 50.4 (1999), pp. 422–428.

Chapter 2

THESIS ORGANIZATION

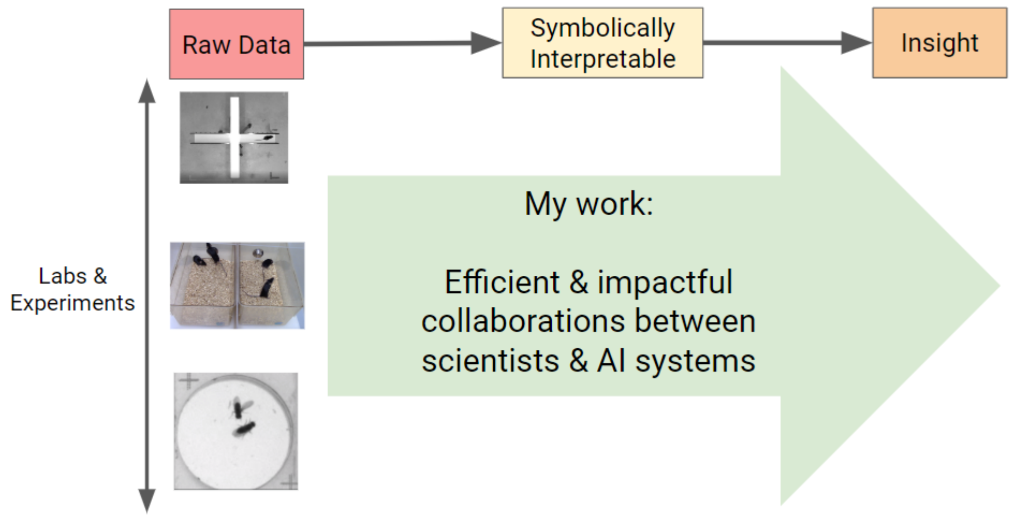

Raw Data
Symbolically Interpretable
Insight
Labs & Experiments
My work:
Efficient & impactful collaborations between scientists & AI systems

Figure 2.1: This thesis will present developing AI for scientists in two main parts: Part II focuses on extracting representations from raw data, and Part III focuses on transforming these representations to insight by integrating symbolic domain knowledge. In the concluding section (Part IV), we will discuss how these systems could be connected end-to-end.

Part II: Representation & Structural Discovery from Real-World Data

Part II will focus on methods that transform data to lower-dimensional representations and structures that improves performance for downstream analysis. These methods are generally learned in a self-supervised way, such that we do not require human annotations which are generally expensive to obtain on scientific datasets.

In Chapter 4, we discuss methods for keypoint discovery from video data that works directly on a range of organisms, from mice, to flies, to humans. Our method discovers locations that encode information to reconstruct agent movements. We show that this idea works in 2D on monocular videos, as well as in 3D on multi-view videos. In Chapter 5, we study pose representations, and in particular, how to map a set of 2D keypoints to representations that are invariant of camera viewpoints. These pose embeddings are applicable in a range of tasks, including cross-view action recognition, pose retrieval, and video alignment. Evaluation representations is crucial for comparison, and Chapter 6 proposes a new dataset from biology experiments to evaluate behavioral representations. Our dataset is collected across three organisms, and is based on a set of experimentally-defined downstream tasks. We transition towards a discussion of neurosymbolic methods in Chapter 7, which focuses on our work of combining variational autoencoders with neurosymbolic methods. The result is a representation that contains both symbolic components (generated through a DSL) and neural components.

Part III: Integrating Symbolic Domain Knowledge with Learning

Part III will focus on methods that learns from both data and domain knowledge to improve data efficiency and model interpretability. These methods are generally part of the neurosymbolic learning movement, which aims to integrate symbolic knowledge from scientists with the flexibility of neural networks.

In Chapter 9, we study task programming, a framework to unify self-supervision from data and programmatic supervision from domain experts for representation learning. Results demonstrate that our framework reduces annotated data requirements by up to 90% compared to previous methods. The programs in Chapter 9 are hand-crafted by experts, and Chapter 10 discusses NEAR, a method that learns programs automatically from the data and a DSL. NEAR uses neural heuristics to guide the process of learning both the architecture and parameters of differentiable programs. We discuss how program learning can result in inherently interpretable models compared to neural networks. We study how program learning further improves data efficiency in Chapter 11, where these synthesized programs can be used as sources of weak supervision. In addition to weak supervision, we also explore program synthesis and active learning. Chapter 12 is a perspective paper on the opportunities and challenges in neurosymbolic learning and science. Our discussion is grounded in the behavior analysis domain, and we outline the potential for future work at this intersection.

Part IV: Conclusion

Finally, we will conclude with a discussion of future works on interactivity and expert-in-the-loop systems in science, extending our framework to other domains, and further connecting the science and AI research communities. The development of these frameworks requires a collaborative process between ML researchers and labs with diverse domain knowledge, experimental setups, and data. Through this process, my research lays the groundwork towards a general scientist-in-the-loop framework that learns from expert knowledge and experimental data across domains. This framework has the potential to unify analysis efforts and share insights broadly across science.

Part II

Representation & Structural Discovery from Real-World Data

Chapter 3

OVERVIEW

Scientific inquiries rely on semantically-meaningful structure extracted from raw data. For example, insights on the relationship between brain and behavior are based on body parts (keypoints) or behavioral categories [2] computed from raw video (Figure 2). However, scientists find manual data annotation of these structures to be time-consuming and tedious, and are asking for algorithms that reveal structures and features in raw data to guide their intepretation. My approach develops methods that automatically discover these structures, leveraging properties of experimental data not well-explored in ML, such as known capture conditions [5], spatiotemporal relationships [6], or connections between data streams [4].

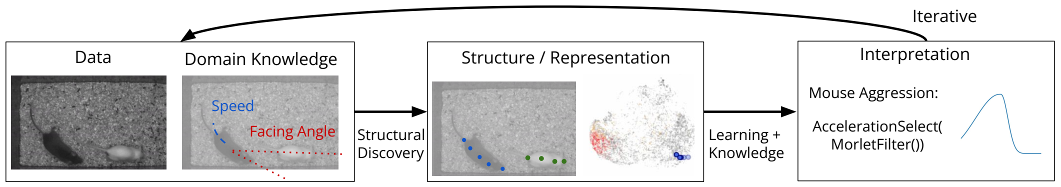


Figure 3.1: We will discuss methods that learn from video data and expert knowledge to discover structure and interpretable quantitative representations of animal behavior.

My work studies new methods for self-supervised learning and representation learning on experimental data. In this process, we ultimately need to make assumptions on our data and desired representations. There are a few ways that we encode prior knowledge in order to discover useful representations:

- The design of the model architecture: The model architecture is crucial, since it affects the embedding dimensionality, its structure, and properties (for example, CNNs assume translational equivariance). The model architecture need to take into account structure of the data, downstream tasks, as well as amount of available data. There is often a tradeoff between the size of the hypothesis space represented by the model, to the amount of data needed to train the model. Techniques such as transfer learning and fine-tuning could help reduce these requirements.

- The design of loss functions: The loss function is another important choice in representation learning — the model is trained to optimize this function. Besides the architecture, we can also use the loss function to encourage the model to learn useful representations (for example, using contrastive loss to learn invariances). Typically, models will only be able to reach local minima and may not always be able to converge. There is often a tradeoff between the complexity of the loss function, to how easy the model is to optimize.

- The design of datasets: The data that the model is trained and evaluated on is another way to encode prior knowledge. This data is often split into train, validation, and test. Out of these splits, it is crucial that the test dataset be designed to be representative of the real-world task where the model will be deployed. For training data, synthetic data and other types of data augmentation are commonly used to artificially increase the size of the training data. The training data encodes prior knowledge into the model since the loss function is optimized on this dataset, and often any biases in the training data is passed onto the model.

In general, not only has my work improved data efficiency of existing categories [4, 3, 1, 8], I have also developed methods for discovering meaningful representations [6, 7]. These representations benefit a range of downstream tasks, such as [4] which improves the performance of pose retrieval, video alignment, and action recognition; and [6] for imputing missing brain recordings for neural decoding in neuroscience, thus enabling new experimental design.

Each chapter in this part covers a different flavor of representation learning:

- Chapter 4: We develop a new loss function that leverages properties from experimental data (animal behavioral videos) alongside a geometric bottleneck in the model to discover 2D and 3D keypoints.

- Chapter 5: We learn a view-invariant representation of 2D pose data by using a loss that encourages the model to map 2D poses with similar 3D poses closer together in the representation space, and different 3D poses further apart.

- Chapter 6: We evaluate trajectory and video representations of animal behavior on a suite of downstream tasks, from both experimentally-derived labels as well as human annotations.

- Chapter 7: We explore the variational autoencoder method alongisde a neurosymbolic encoder, to learn both a symbolic and neural representation in an unsupervised way.

References

[1] Ting Liu*, Jennifer J. Sun*, Long Zhao, Jiaping Zhao, Liangzhe Yuan, Yuxiao Wang, Liang-Chieh Chen, Florian Schroff, and Hartwig Adam. "View-Invariant, Occlusion-Robust Probabilistic Embedding for Human Pose." In: *International Journal of Computer Vision* 130.1 (2022), pp. 111–135. URL: `https://arxiv.org/pdf/2010.13321.pdf`.

[2] Cristina Segalin, Jalani Williams, Tomomi Karigo, May Hui, Moriel Zelikowsky, Jennifer J. Sun, Pietro Perona, David J. Anderson, and Ann Kennedy. "The Mouse Action Recognition System (MARS) Software Pipeline for Automated Analysis of Social Behaviors in Mice." In: *eLife* 10 (2021), e63720.

[3] Jennifer J. Sun, Ann Kennedy, Eric Zhan, David J. Anderson, Yisong Yue, and Pietro Perona. "Task Programming: Learning Data Efficient Behavior Representations." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 2876–2885. URL: `https://arxiv.org/pdf/2011.13917.pdf`.

[4] Jennifer J. Sun, Jiaping Zhao, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, and Ting Liu. "View-Invariant Probabilistic Embedding for Human Pose." In: *European Conference on Computer Vision (ECCV)* (2020), pp. 53–70. URL: `https://arxiv.org/pdf/1912.01001.pdf`.

[5] Jennifer J. Sun*, Serim Ryou*, Roni H. Goldshmid, Brandon Weissbourd, John O. Dabiri, David J. Anderson, Ann Kennedy, Yisong Yue, and Pietro Perona. "Self-Supervised Keypoint Discovery in Behavioral Videos." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 2171–2180. URL: `https://arxiv.org/pdf/2112.05121.pdf`.

[6] Sabera Talukder*, Jennifer J. Sun*, Matthew Leonard, Bingni W. Brunton, and Yisong Yue. "Deep Neural Imputation: A Framework for Recovering Incomplete Brain Recordings." In: *Learning from Time Series for Health Workshop at Neural Information Processing Systems (NeurIPS)* (2022).

[7] Eric Zhan*, Jennifer J. Sun*, Ann Kennedy, Yisong Yue, and Swarat Chaudhuri. "Unsupervised Learning of Neurosymbolic Encoders." In: *Transactions on Machine Learning Research* (2022). URL: `https://arxiv.org/pdf/2107.13132.pdf`.

[8] Long Zhao, Yuxiao Wang, Jiaping Zhao, Liangzhe Yuan, Jennifer J. Sun, Florian Schroff, Hartwig Adam, Xi Peng, Dimitris Metaxas, and Ting

Liu. “Learning View-Disentangled Human Pose Representation by Contrastive Cross-View Mutual Information Maximization.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 12793–12802.

Chapter 4

DISCOVERING KEYPOINTS IN 2D AND 3D

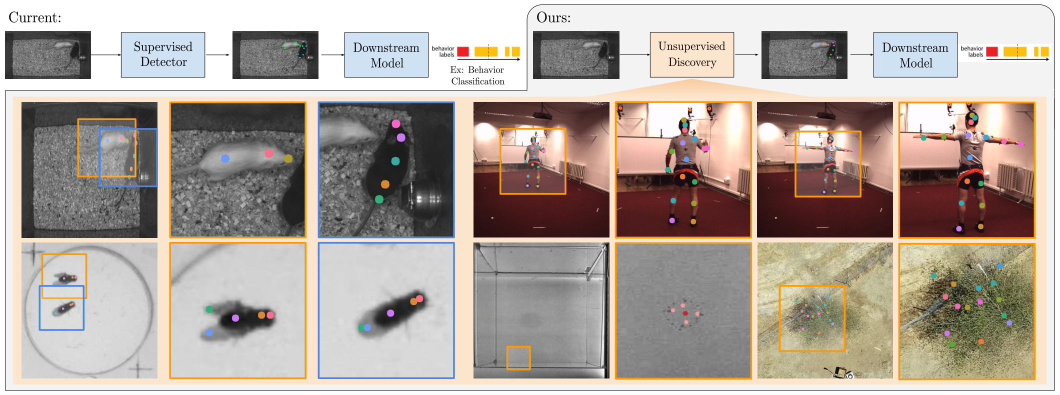


Figure 4.1: We propose a method to discover keypoints from behavioral videos without the need for manual keypoint or bounding box annotations.

This chapter is mainly based on the following papers:

[1] Jennifer J. Sun*, Lili Karashchuk*, Amil Dravid*, Serim Ryou, Sonia Fereidooni, John C Tuthill, Aggelos Katsaggelos, Bingni W Brunton, Georgia Gkioxari, Ann Kennedy, et al. "BKinD-3D: Self-Supervised 3D Keypoint Discovery from Multi-View Videos." In: (2023), pp. 9001–9010. URL: `https://arxiv.org/pdf/2212.07401.pdf`.

[2] Jennifer J. Sun*, Serim Ryou*, Roni H. Goldshmid, Brandon Weissbourd, John O. Dabiri, David J. Anderson, Ann Kennedy, Yisong Yue, and Pietro Perona. "Self-Supervised Keypoint Discovery in Behavioral Videos." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 2171–2180. URL: `https://arxiv.org/pdf/2112.05121.pdf`.

Abstract. We propose a method for learning the posture and structure of agents from unlabelled behavioral videos. Starting from the observation that behaving agents are generally the main sources of movement in behavioral videos, our method, Behavioral Keypoint Discovery (B-KinD), uses an encoder-decoder architecture with a geometric bottleneck to reconstruct the spatiotemporal difference between video frames. By focusing only on regions of movement, our approach works directly on

input videos without requiring manual annotations. Experiments on a variety of agent types (mouse, fly, human, jellyfish, and trees) demonstrate the generality of our approach and reveal that our discovered keypoints represent semantically meaningful body parts, which achieve state-of-the-art performance on keypoint regression among self-supervised methods. Additionally, B-KinD achieve comparable performance to supervised keypoints on downstream tasks, such as behavior classification, suggesting that our method can dramatically reduce model training costs vis-a-vis supervised methods.

4.1 Introduction

Automatic recognition of object structure, for example in the form of keypoints and skeletons, enables models to capture the essence of the geometry and movements of objects. Such structural representations are more invariant to background, lighting, and other nuisance variables and are much lower-dimensional than raw pixel values, making them good intermediates for downstream tasks, such as behavior classification [3, 17, 53], video alignment [59, 35], and physics-based modeling [6, 55].

However, obtaining annotations to train supervised pose detectors can be expensive, especially for applications in behavior analysis. For example, in behavioral neuroscience [45], datasets are typically small and lab-specific, and the training of a custom supervised keypoint detector presents a significant bottleneck in terms of cost and effort. Additionally, once trained, supervised detectors often do not generalize well to new agents with different structures without new supervision. The goal of our work is to enable keypoint discovery on new videos without manual supervision, in order to facilitate behavior analysis on novel settings and different agents.

Recent unsupervised/self-supervised methods have made great progress in keypoint discovery [28, 29, 71], but these methods are generally not designed for behavioral videos. In particular, existing methods do not address the case of multiple and/or non-centered agents, and often require inputs as cropped bounding boxes around the object of interest, which would require an additional detector module to run on real-world videos. Furthermore, these methods do not exploit relevant structural properties in behavioral videos (e.g., the camera and the background are typically stationary, as observed in many real-world behavioral datasets [53, 17, 4, 38, 45, 30]).

To address these challenges, the key to our approach is to discover keypoints based on reconstructing the *spatiotemporal difference* between video frames. Inspired by previous works based on image reconstruction [29, 50], we use an encoder-decoder setup to encode input frames into a geometric bottleneck, and train the model for reconstruction. We then use spatiotemporal difference as a novel reconstruction target for keypoint discovery, instead of single image reconstruction. Our method enables the model to focus on discovering keypoints on the behaving agents, which are generally the only source of motion in behavioral videos.

Our self-supervised approach, **B**ehavioral **K**eypo**in**t **D**iscovery (B-KinD), works without manual supervision across diverse organisms (Figure 3.1). Results show that our discovered keypoints achieve state-of-the-art performance on downstream tasks among other self-supervised keypoint discovery methods. We demonstrate the performance of our keypoints on behavior classification [58], keypoint regression [29], and physics-based modeling [6]. Thus, our method has the potential for transformative impact in behavior analysis: first, one may discover keypoints from behavioral videos for new settings and organisms; second, unlike methods that predict behavior directly from video, our low-dimensional keypoints are semantically meaningful so that users can directly compute behavioral features; finally, our method can be applied to videos without the need for manual annotations.

Contributions

1. Self-supervised method for discovering keypoints from real-world behavioral videos, based on spatiotemporal difference reconstruction.
2. Experiments across a range of organisms (mice, flies, human, jellyfish, and tree) demonstrating the generality of the method and showing that the discovered keypoints are semantically meaningful.
3. Quantitative benchmarking on downstream behavior analysis tasks showing performance that is comparable to supervised keypoints.

4.2 Related Work

Analyzing Behavioral Videos

Video data collected for behavioral experiments often consists of moving agents recorded from stationary cameras [1, 17, 4, 43, 14, 30]. These behavioral videos contain different model organisms studied by researchers, such as fruit flies [17, 32, 3] and mice [23, 53, 30]. From these recorded video data, there has been an increasing effort to automatically estimate poses of agents and classify behavior [32, 23, 18, 16, 53].

Pose estimation models that were developed for behavioral videos [40, 20, 53, 46] require human annotations of anatomically defined keypoints, which are expensive and time-consuming to obtain. In addition to the cost, not all data can be crowd-sourced due to the sensitive nature of some experiments. Furthermore, organisms that are translucent (jellyfish) or with complex shapes (tree) can be difficult for non-expert humans to annotate. Our goal is to enable keypoint discovery on videos for behavior analysis, without the need for manual annotations.

Keypoint Estimation

Keypoint estimation models aim to localize a predefined set of keypoints from visual data, and many works in this area focus on human pose. With the success of fully convolutional neural networks [54], recent methods [42, 70, 11, 63] employ encoder-decoder networks by predicting high-resolution outputs encoded with 2D Gaussian heatmaps representing each part. To improve model performance, [42, 70, 63] propose an iterative refinement approach, [11, 49] design efficient learning signals, and [12, 68] exploit multi-resolution information. Similar to these works, we also use 2D Gaussian heatmaps to represent parts as keypoints, but instead of using human-defined keypoints, we aim to discover keypoints from video data without manual supervision.

Unsupervised Part Discovery

Though keypoints provide a useful tool for behavior analysis, collecting annotations is time-consuming and labor-intensive especially for new domains that have not been previously studied. Unsupervised keypoint discovery [29, 71, 28] has been proposed to reduce keypoint annotation effort and there have been many promising results on centered and/or aligned objects, such as facial images and humans with an upright pose. These methods train and evaluate on images where the object of interest is centered in an input bounding box. Most of the approaches [71, 29, 36] use an autoencoder-based architecture to disentangle the appearance and geometry representation for the image reconstruction task. Our setup is similar in that we also use an encoder-decoder architecture, but crucially, we reconstruct spatiotemporal difference between video frames, instead of the full image as in previous works. We found that this enables our discovered keypoints to track semantically-consistent parts without manual supervision, requiring neither keypoints nor bounding boxes.

There are also works for parts discovery that employ other types of supervision [28,

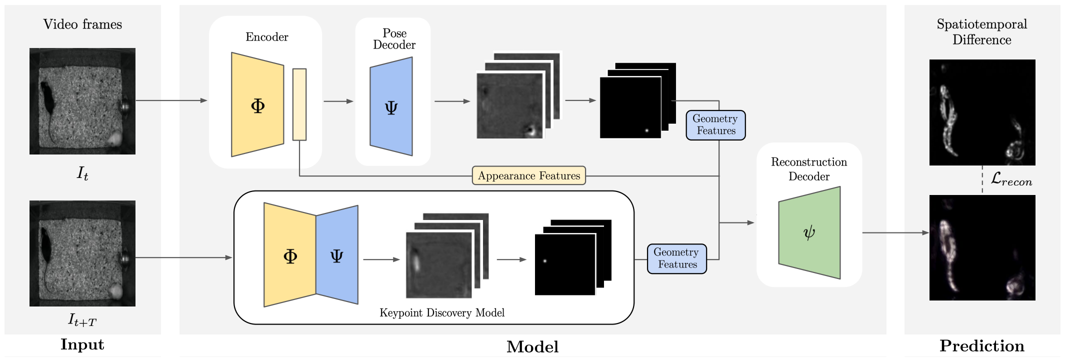


Figure 4.2: BKinD, an approach for keypoint discovery from spatiotemporal difference reconstruction. I_t and I_{t+T} are video frames at time t and $t+T$. Both frame I_t and frame I_{t+T} are fed to an appearance encoder Φ and a pose decoder Ψ. Given the appearance feature from I_t and geometry features from both I_t and I_{t+T} (Sec 4.3), our model reconstructs the spatiotemporal difference (Sec 4.3) computed from two frames using the reconstruction decoder ψ.

52, 50]. For example, [50] proposed a weakly-supervised approach using class label to discriminate parts to handle viewpoint changes, [28] incorporated pose prior obtained from unpaired data from different datasets in the same domain, and [52] proposed a template-based geometry bottleneck based on a pre-defined 2D Gaussian-shaped template. Different from these approaches, our method does not require any supervision beyond the behavioral videos. We chose to focus on this setting since other supervisory sources are not readily available for emerging domains (for example, jellyfish, trees).

4.3 2D Keypoint Discovery

The goal of BKinD (Figure 4.2) is to discover semantically meaningful keypoints in behavioral videos of diverse organisms without manual supervision. We use an encoder-decoder setup similar to previous methods [29, 50], but instead of image reconstruction, here we study a novel reconstruction target based on spatiotemporal difference. In behavioral videos, the camera is generally fixed with respect to the world, such that the background is largely stationary and the agents (e. g. mice moving in an enclosure) are the only source of motion. Thus spatiotemporal differences provide a strong cue to infer location and movements of agents.

Self-supervised keypoint discovery setup

Given a behavioral video, our work aims to reconstruct regions of motion between a reference frame I_t (the video frame at time t) and a future frame I_{t+T} (the video frame T timesteps later, for some set value of T.) We accomplish this by extracting appearance features from frame I_t and keypoint locations ("geometry features") from both frames I_t and I_{t+T} (Figure 4.2). In contrast, previous works [29, 36, 28, 50, 52] use appearance features from I_t and geometry features from I_{t+T} to reconstruct the full image I_{t+T} (instead of difference between I_t and I_{t+T}).

We use an encoder-decoder architecture, with shared appearance encoder Φ, geometry decoder Ψ, and reconstruction decoder ψ. During training, the pair of frames I_t and I_{t+T} are fed to the appearance encoder Φ to generate appearance features, and those features are then fed into the geometry decoder Ψ to generate geometry features. In our approach, the reference frame I_t is used to generate both appearance and geometry representations, and the future frame I_{t+T} is only used to generate a geometry representation. The appearance feature h_a^t for frame I_t are defined simply as the output of Φ: $h_a^t = \Phi(I_t)$.

The pose decoder Ψ outputs K raw heatmaps $\mathbf{X}_i \in \mathbb{R}^2$, then applies a spatial softmax operation on each heatmap channel. Given the extracted $p_i = (u_i, v_i)$ locations for $i = \{1, \ldots, K\}$ keypoints from the spatial softmax, we define the geometry features h_g^t to be a concatenation of 2D Gaussians centered at (u_i, v_i) with variance σ.

Finally, the concatenation of the appearance feature h_a^t and the geometry features h_g^t and h_g^{t+T} is fed to the decoder ψ to reconstruct the learning objective $\hat{S}$ discussed in the next section: $\hat{S} = \psi(h_a^t, h_g^t, h_g^{t+T})$.

Learning formulation

Spatiotemporal difference. Our method works with different types of spatiotemporal differences as reconstruction targets. For example:

1. Structural Similarity Index Measure (SSIM) [69]. This is a method for measuring the perceived quality of the two images based on luminance, contrast, and structure features. To compute our reconstruction target based on SSIM, we apply the SSIM measure locally on corresponding patches between I_t and I_{t+T} to build a similarity map between frames. Then we compute dissimilarity by taking the negation of the similarity map.

2. Frame differences. When the video background is static with little noise, simple frame differences, such as absolute difference ($S_{|d|} = |I_{t+T} - I_t|$) or raw difference ($S_d = I_{t+T} - I_t$), can also be directly applied as a reconstruction target.

Reconstruction loss. We apply perceptual loss [31] for reconstructing the spatiotemporal difference S. Perceptual loss compares the L2 distance between the features computed from VGG network ϕ [56]. The reconstruction $\hat{S}$ and the target S are fed to VGG network, and mean squared error is applied to the features from the intermediate convolutional blocks:

$$\mathcal{L}_{recon} = \left\|\phi(S(I_t, I_{t+T})) - \phi(\hat{S}(I_t, I_{t+T}))\right\|_2 . \tag{4.1}$$

Rotation equivariance loss. In cases where agents can move in many directions (e.g., mice filmed from above can translate and rotate freely), we would like our keypoints to remain semantically consistent. We enforce rotation-equivariance in the discovered keypoints by rotating the image with different angles and imposing that the predicted keypoints should move correspondingly. We apply the rotation equivariance loss (similar to the deformation equivariance in [64]) on the generated heatmap.

Given reference image I and the corresponding geometry bottleneck h_g, we rotate the geometry bottleneck to generate pseudo labels $h_g^{R^\circ}$ for rotated input images I^{R° with degree $R = \{90^\circ, 180^\circ, 270^\circ\}$. We apply mean squared error between the predicted geometry bottlenecks $\hat{h_g}$ from the rotated images and the generated pseudo labels h_g:

$$\mathcal{L}_r = \left\|h_g^{R^\circ} - \hat{h}_g(I^{R^\circ})\right\|_2 . \tag{4.2}$$

Separation loss. Empirical results show that rotation equivariance encourages the discovered keypoints to converge at the center of the image. We apply separation loss to encourage the keypoints to encode unique coordinates, and prevent the discovered keypoints from being centered at the image coordinates [71]. The separation loss is defined as follows:

$$\mathcal{L}_s = \sum_{i \neq j} \exp\left(\frac{-(p_i - p_j)^2}{2\sigma_s^2}\right). \tag{4.3}$$

Final objective. Our final loss function is composed of three parts: reconstruction loss $\mathcal{L}_{recon}$, rotation equivariance loss $\mathcal{L}_r$, and separation loss $\mathcal{L}_s$:

$$\mathcal{L} = \mathcal{L}_{recon} + \mathbb{1}_{epoch>n}(w_r\mathcal{L}_r + w_s\mathcal{L}_s). \tag{4.4}$$

We adopt curriculum learning [2] and apply $\mathcal{L}_r$ and $\mathcal{L}_s$ once the keypoints are consistently discovered from the semantic parts of the target instance.

Feature extraction for behavior analysis

Following standard approaches [53, 4, 23], we use the discovered keypoints from BKinD as input to a behavior quantification module: either supervised classifiers or

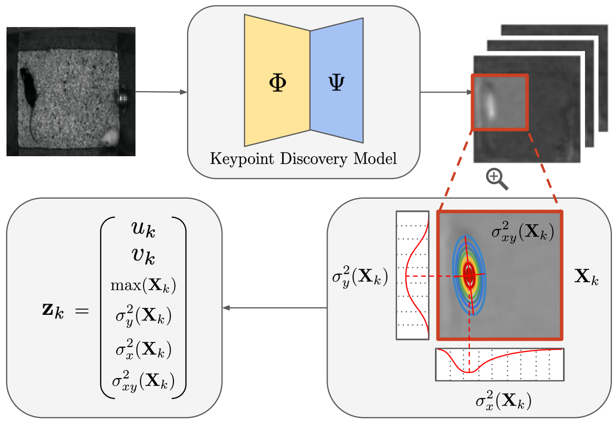

Φ
Ψ
Keypoint Discovery Model
$\mathbf{z}_k = \begin{pmatrix} u_k \\ v_k \\ \max(\mathbf{X}_k) \\ \sigma_y^2(\mathbf{X}_k) \\ \sigma_x^2(\mathbf{X}_k) \\ \sigma_{xy}^2(\mathbf{X}_k) \end{pmatrix}$
$\sigma_{xy}^2(\mathbf{X}_k)$
$\sigma_y^2(\mathbf{X}_k)$
$\mathbf{X}_k$
$\sigma_x^2(\mathbf{X}_k)$

Figure 4.3: Extracting information from the raw heatmap (Section 4.3): the confidence scores and the covariance matrices are computed from normalized heatmaps. Note that the features are computed for all x, y coordinates. We visualize the zoomed area around the target instance for illustrative purposes.

a physics-based model. Note that this is a separate process from keypoint discovery; we feed discovered geometry information into a downstream model.

In addition to discovered keypoints, we extracted additional features from the raw heatmap (Figure 4.3) to be used as input to our downstream modules. For instance, we found that the confidence and the shape information from the of the network prediction of keypoint location was informative. When a target part is well localized, our keypoint discovery network produces a heatmap with a single high peak with low variance; conversely, when a target part is occluded, the raw heatmap contains a blurred shape with lower peak value. This "confidence" score (heatmap peak value) is also a good indicator for whether keypoints are discovered on the background (blurred over the background with low confidence) or tracking anatomical body parts (peaked with high confidence), visualized in Supplementary materials of [62]. The shape of a computed heatmap can also reflect shape information of the target (e.g., stretching).

Given a raw heatmap $\mathbf{X}_k$ for part k, the confidence score is obtained by choosing the maximum value from the heatmap, and the shape information is obtained by computing the covariance matrix from the heatmap. Figure 4.3 visualizes the features we extract from the raw heatmaps. Using the normalized heatmap as the probability distribution, additional geometric features are computed:

$$\begin{aligned}\sigma_x^2(\mathbf{X}_k) &= \sum_{ij}(x_i - u_k)^2\mathbf{X}_k(i,j),\\ \sigma_y^2(\mathbf{X}_k) &= \sum_{ij}(y_j - v_k)^2\mathbf{X}_k(i,j),\\ \sigma_{xy}^2(\mathbf{X}_k) &= \sum_{ij}(x_i - u_k)(y_j - v_k)\mathbf{X}_k(i,j).\end{aligned} \tag{4.5}$$

4.4 Experiments for 2D

We demonstrate that B-KinD is able to discover consistent keypoints in real-world behavioral videos across a range of organisms. We evaluate our keypoints on downstream tasks for behavior classification and pose regression, then illustrate additional applications of our keypoints.

Datasets

CalMS21. CalMS21 [58] is a large-scale dataset for behavior analysis consisting of videos and trajectory data from a pair of interacting mice. Every frame is annotated by an expert for three behaviors: sniff, attack, mount. There are 507k frames in the train split, and 262k frames in the test split (video frame: 1024×570, mouse: approx 150×50). We use only the train split on videos without miniscope cable to train B-KinD. Following [58], the downstream behavior classifier is trained on the entire training split, and performance is evaluated on the test split.

MARS-Pose. This dataset consists of a set of videos with similar recording conditions to the CalMS21 dataset. We use a subset of the MARS pose dataset [53] with keypoints from manual annotations to evaluate the ability of our model to predict human-annotated keypoints, with $\{10, 50, 100, 500\}$ images for train and 1.5k images for test.

Fly vs. Fly. These videos consists of interactions between a pair of flies, annotated per frame by domain experts. We use the Aggression videos from the Fly vs. Fly dataset [17], with the train and test split having 1229k and 322k frames, respectively (video frame: 144×144, fly: approximately 30×10). Similar to [57], we evaluate on behaviors of interest with more than 1000 frames in the training set (lunge, wing threat, tussle).

Human3.6M. Human3.6M [24] is a large-scale motion capture dataset, which consists of 3.6 million human poses and images from 4 viewpoints. To quantitatively measure the pose regression performance against baselines, we use the Simplified

Human3.6M dataset, which consists of 800k training and 90k testing images with 6 activities in which the human body is mostly upright. We follow the same evaluation protocol from [71] to use subjects 1, 5, 6, 7, and 8 for training and 9 and 11 for testing. We note that each subject has different appearance and clothing.

Jellyfish. The jellyfish data is an in-house video dataset containing 30k frames of recorded swimming jellyfish (video frame: 928×1158, jellyfish: approximately 50 pix in diameter). We use this dataset to qualitatively test the performance of B-KinD on a new organism, and apply our keypoints to detect the pulsing motion of the jellyfish.

Vegetation. This is an in-house dataset acquired over several weeks using a drone to record the motion of swaying trees. The dataset consists of videos of an oak tree and corresponding wind speeds recorded using an anemometer, with a total of 2.41M video frames (video frame: 512×512, oak tree: varies, approximately $\frac{1}{4}$ of the frame). We evaluate this dataset using a physics-based model [6] that relates the visually observed oscillations to the average wind speeds.

Training and evaluation procedure

We train B-KinD using the full objective in Section 4.3. During training, we rescale images to 256×256 and use T of around 0.2 seconds, except Human3.6M, where we use 128×128. Unless otherwise specified, all experiments are ran with all keypoints discovered from B-KinD with SSIM reconstruction and with 10 keypoints for mouse, fly, and jellyfish, 16 keypoints for Human3.6M, 15 keypoints for Vegetation. We train on the train split of each dataset as specified, except for jellyfish and vegetation, where we use the entire dataset. Additional details are in the Supplementary materials of [62].

After training the keypoint discovery model, we extract the keypoints and use it for different evaluations based on the labels available in the dataset: behavior classification (CalMS21, Fly), keypoint regression (MARS-Pose, Human), and physics-based modeling (Vegetation).

For keypoint regression, similar to previous works [29, 28], we compare our regression with a fully supervised 1-stack hourglass network [42]. We evaluate keypoint regression on Simplified Human 3.6M by using a linear regressor without a bias term, following the same evaluation setup from previous works [71, 36]. On MARS-Pose, we train our model in a semi-supervised fashion with $10, 50, 100, 500$ supervised keypoints to test data efficiency. For behavior classification, we evaluate

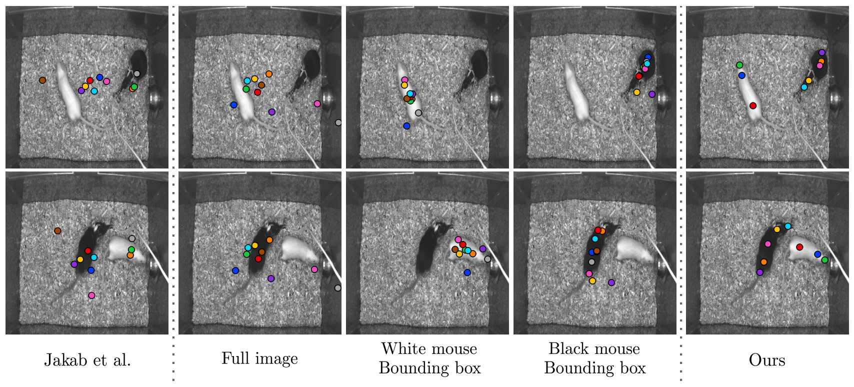


Figure 4.4: Comparison with existing methods [29], full image, bounding box, and SSIM reconstruction (ours). "Jakab et al. " [29] and "full image" results are based on full image reconstruction. "White mouse bounding box" and "black mouse bounding box" show the results when the cropped bounding boxes were fed to the network for image reconstruction.

on CalMS21 and Fly, using available frame-level behavior annotations. To train behavior classifiers, we use the specified train split of each dataset. For CalMS21 and Fly, we train the 1D Convolutional Network benchmark model provided by [58] using B-KinD keypoints. We evaluate using mean average precision (MAP) weighted equally over all behaviors of interest.

Behavior classification results

CalMS21 Behavior Classification. We evaluate the effectiveness of B-KinD for behavior classification (Table 4.1). Compared to supervised keypoints trained for this task, our keypoints (without manual supervision) is comparable when using both pose and confidence as input. Compared to other self-supervised methods, even those that use bounding boxes, our discovered keypoints on the full image generally achieve better performance.

Keypoints discovered with image reconstruction, similar to baselines [29, 50] cannot track the agents well without using bounding box information (Figure 4.4) and does not perform well for behavior classification (Table 4.1). When we provide bounding box information to the model based on image reconstruction, the performance is significantly improved, but this model does not perform as well as B-KinD keypoints from spatiotemporal difference reconstruction.

For the per-class performance (see the Supplementary materials of [62]), the biggest gap exists between B-KinD and MARS on the "attack" behavior. This is likely

CalMS21	Pose	Conf	Cov	MAP
Fully supervised				
MARS † [53]	✓			.856 ± .010
	✓	✓		.874 ± .003
	✓	✓	✓	.880 ± .005
Self-supervised				
Jakab et al. [29]	✓			.186 ± .008
Image Recon.	✓			.182 ± .007
	✓	✓		.184 ± .006
	✓	✓	✓	.165 ± .012
Image Recon. bbox†	✓			.819 ± .008
	✓	✓		.812 ± .006
	✓	✓	✓	.812 ± .010
Ours	✓			.814 ± .007
	✓	✓		.857 ± .005
	✓	✓	✓	.852 ± .013

Table 4.1: Behavior Classification Results on CalMS21. "Ours" represents classifiers using input keypoints from our discovered keypoints. "conf" represents using the confidence score, and "cov" represents values from the covariance matrix of the heatmap. † refers to models that require bounding box inputs before keypoint estimation. Mean and std dev from 5 classifier runs are shown.

Fly	MAP
Hand-crafted features	
FlyTracker [17]	.809 ± .013
Self-supervised + generic features	
Image Recon.	.500 ± .024
Image Recon. bbox†	.750 ± .020
Ours	.727 ± .022

Table 4.2: Behavior Classification Results on Fly. "FlyTracker" represents classifiers using hand-crafted inputs from [17]. The self-supervised keypoints all use the same "generic features" computed on all keypoints: speed, acceleration, distance, and angle. † refers to models that require bounding box inputs before keypoint estimation. Mean and std dev from 5 classifier runs are shown.

because during attack, the mice are moving quickly, and there exists a lot of motion blur and occlusion which is difficult to track without supervision. However, once we extract more information from the heatmap, through computing keypoint confidence, our keypoints perform comparably to MARS.

Fly Behavior Classification. The FlyTracker [17] uses hand-crafted features computed from the image, such as contrast, as well as features from tracked fly body parts, such as wing angle or distance between flies. Using discovered keypoints,

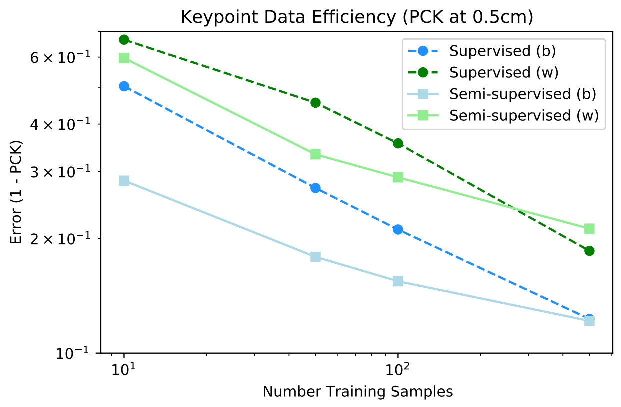


Figure 4.5: Keypoint data efficiency on MARS-Pose. The supervised model is based on [53] using stacked hourglass [42], while the semi-supervised model uses both our self-supervised loss and supervision. PCK is computed at $0.5cm$ threshold, averaged across nose, ears, and tail keypoints, over 3 runs. “b” and “w” indicates the black and white mouse, respectively.

we compute comparable features without assuming keypoint identity, by computing speed and acceleration of every keypoint, distance between every pair, and angle between every triplet. For all self-supervised methods, we use keypoints, confidence, and covariance for behavior classification. Results demonstrate that while there is a small gap in performance to the supervised estimator, our discovered keypoints perform much better than image reconstruction, and is comparable to models that require bounding box inputs (Table 4.2).

Pose regression results

MARS Pose Regression. We evaluate the pose estimation performance of our method in the setting where some human annotated keypoints exist (Figure 4.5). For this experiment, we train B-KinD in a semi-supervised fashion, where the loss is a sum of both our keypoint discovery objective (Section 4.3) as well as standard keypoint estimation objectives based on MSE [53]. For both black and white mouse, when using our keypoint discovery objective in a semi-supervised way during training, we are able to track keypoints more accurately compared to the supervised method [53] alone. We note that the performance of both methods converge at around 500 annotated examples.

Simplified Human 3.6M Pose Regression. To compare with existing keypoint discovery methods, we evaluate our discovered keypoints on Simplified Human3.6M (a standard benchmarking dataset) by regressing to annotated keypoints (Table 4.3).

Simplified H36M	all	wait	pose	greet	direct	discuss	walk
Fully supervised:							
Newell [42]	2.16	1.88	1.92	2.15	1.62	1.88	2.21
Self-supervised + unpaired labels							
Jakab [28]‡	2.73	2.66	2.27	2.73	2.35	2.35	4.00
Self-supervised + template							
Schmidtke [52]	3.31	3.51	3.28	3.50	3.03	2.97	3.55
Self-supervised + regression							
Thewlis [64]	7.51	7.54	8.56	7.26	6.47	7.93	5.40
Zhang [71]	4.14	5.01	4.61	4.76	4.45	4.91	4.61
Lorenz [36]	2.79	–	–	–	–	–	–
Ours (best)	2.44	2.50	2.22	2.47	2.22	2.77	2.50
Ours (mean)	2.53	2.58	2.31	2.56	2.34	2.83	2.58
Ours (std)	.056	.047	.062	.048	.066	.048	.063

Table 4.3: Comparison with state-of-the-art methods for landmark prediction on Simplified Human 3.6M. The error is in %-MSE normalized by image size. All methods predict 16 keypoints except for [28]‡, which uses 32 keypoints for training a prior model from the Human 3.6M dataset. B-Kind results are computed from 5 runs.

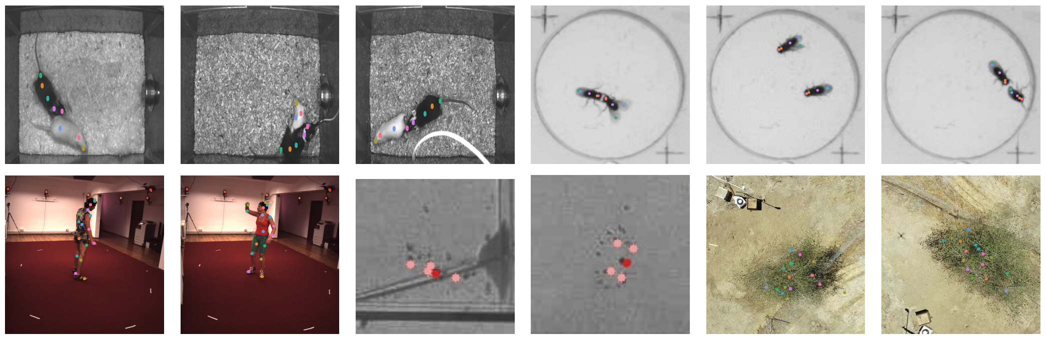


Figure 4.6: Qualitative Results of BKinD. Qualitative results for BKinD trained on CalMS21 (mouse), Fly vs. Fly (fly), Human3.6M (human), Jellyfish and Vegetation (tree). Additional visualizations are in the Supplementary materials of [62].

Though our method is directly applicable to full images, we train the discovery model using cropped bounding box for a fair comparison with baselines, which all use cropped bounding boxes centered on the subject. Compared to both self-supervised + prior information and self-supervised + regression, our method shows state-of-the-art performance on the keypoint regression task, suggesting spatiotemporal difference is an effective reconstruction target for keypoint discovery.

Additional applications

We show qualitative performance and demonstrate additional downstream tasks using our discovered keypoints, on pulse detection for Jellyfish and on wind speed regression for Vegetation.

Qualitative Results. Qualitative results (Figure 4.6) demonstrates that BKinD is able to track some body parts consistently, such as the nose of both mice and keypoints along the spine; the body and wings of the flies; the mouth and gonads of the jellyfish; and points on the arms and legs of the human. For visualization only, we show only keypoints discovered with high confidence values (Section 4.3); for all other experiments, we use all discovered keypoints.

Pulse Detection. Jellyfish swimming is among the most energetically efficient forms of transport, and its control and mechanics are studied in hydrodynamics research [13]. Of key interest is the relationship between body plan and swim pulse frequency across diverse jellyfish species. By computing distance between B-KinD keypoints, we are able to extract a frequency spectrogram to study jellyfish pulsing, with a visible band at the swimming frequency (Supplementary materials of [62]). This provides a way to automatically annotate swimming behavior, which could be quickly applied to video from multiple species to characterize the relationship between swimming dynamics and body plan.

Wind Speed Modeling. Measuring local wind speed is useful for tasks such as tracking air pollution and weather forecasting [5]. Oscillations of trees encode information on wind conditions, and as such, videos of moving trees could function as wind speed sensors [5, 6]. Using the Vegetation dataset, we evaluate the ability of our keypoints to predict wind speed using a physics-based model [6]. This model defines the relationship between the mean wind speed and the structural oscillations of the tree, and requires tracking these oscillations from video, which was previously done manually. We show that B-KinD can accomplish this task automatically. Using our keypoints, we are able to regress the measured ground truth wind speed with an $R^2 = 0.79$, suggesting there is a good agreement between the proportionality assumption from [6] and the experimental results using the keypoint discovery model.

Limitations. One issue we did not explore in detail, and which will require further work, is keypoint discovery for agents that may be partially or completely occluded

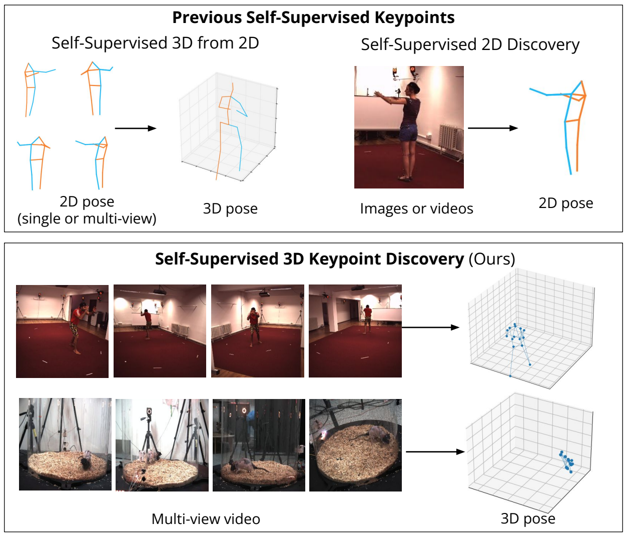

Previous Self-Supervised Keypoints
Self-Supervised 3D from 2D
Self-Supervised 2D Discovery
2D pose (single or multi-view)
3D pose
Images or videos
2D pose
Self-Supervised 3D Keypoint Discovery (Ours)
Multi-view video
3D pose

Figure 4.7: Self-supervised 3D keypoint discovery. Previous work studying self-supervised keypoints either requires 2D supervision for 3D pose estimation or focuses on 2D keypoint discovery (such as in our previous sections). We propose methods for discovering 3D keypoints directly from multi-view videos of different organisms, such as human and rats, without 2D or 3D supervision. The 3D keypoint discovery examples demonstrate the results from our method.

at some point during observation, including self-occlusion. Additionally, similar to other keypoint discovery models [71, 36, 52], we observe left/right swapping of some body parts, such as the legs in a walking human. One approach that might overcome these issues would be to extend our model to discover the 3D structure of the organism, for instance by using data from multiple cameras. Despite these challenges, our model performs comparably to supervised keypoints for behavior classification.

4.5 3D Keypoint Discovery

All animals behave in 3D, and analyzing 3D posture and movement is crucial for a variety of applications, including the study of biomechanics, motor control, and behavior [37]. However, annotations for supervised training of 3D pose estimators are expensive and time-consuming to obtain, especially for studying diverse animal species and varying experimental contexts. Self-supervised keypoint discovery has demonstrated tremendous potential in discovering 2D keypoints from video [29, 28, 62], without the need for manual annotations. These models have not been well-explored in 3D, which is more challenging compared to 2D due to depth ambiguities, a larger search space, and the need to incorporate geometric constraints. Our goal is to enable 3D keypoint discovery of humans and animals from synchronized multi-view videos, without 2D or 3D supervision.

The key to our approach, which we call **B**ehavioral **K**eypo**in**t **D**iscovery in **3D** (BKinD-3D), is to encode self-supervised learning signals from videos across multiple views into a single 3D geometric bottleneck. We leverage the spatiotemporal difference reconstruction loss from our previous section and use multi-view reconstruction to train an encoder-decoder architecture. Our method does not use any bounding boxes or keypoint annotations as supervision. Critically, we impose links between our discovered keypoints to discover connectivity across points. In other words, keypoints on the same parts of the body are connected, so that we are able to enforce joint length constraints in 3D. To show that our model is applicable across multiple settings, we demonstrate our approach on multi-view videos from different organisms.

Related Work in 3D Pose Estimation

There has been a large body of work studying 3D human pose estimation from images or videos, as reviewed in [51, 67], with recent works also focusing on 3D animal poses [15, 37, 19, 33, 21]. Most of these methods are fully supervised from visual data [26, 60, 10], with some models perform lifting starting from 2D poses [39, 8, 44, 47]. We focus our discussion on multi-view 3D pose estimation methods, but all of these models require either 3D or 2D supervision during training. This 2D supervision is typically in the form of pre-trained 2D detectors [34], or ground truth 2D poses [65]. In comparison, our method uses multi-view videos to discover 3D keypoints without 2D or 3D supervision.

Methods more closely related to our work are those that also leverage multi-view structure to estimate 3D pose (Table 4.4). [26] proposed a supervised method that uses learnable triangulation to aggregate 2D information across views to 3D. Here we study similar approaches for representing 3D information, but using self-supervision instead of supervised 3D annotations. Other methods in this space propose training methods such as enforcing consistency of predicted poses across views [48], regression to 3D pose estimated from epipolar geometry of multi-view 2D [34], constraining 3D poses to project to realistic 2D pose [9], or estimates camera parameters using detected and ground truth 2D poses [65]. While we also leverage multi-view information, our goal is different from the work above, in that our approach aims to discover 3D poses without 2D or 3D supervision, given camera parameters.

Method	3D sup.	2D sup.	camera params	data type
Isakov et al. [26] DANNCE [15]	✓	✓	intrinsics extrinsics	real
Rhodin et al. [48]	✓	optional	intrinsics	real
Anipose [33] DeepFly3D [21]	×	✓	intrinsics extrinsics	real
EpipolarPose [34] CanonPose [66]	×	✓	optional	real
MetaPose [65]	×	✓	×	real
Keypoint3D [7]	×	×	intrinsics extrinsics	simulation
Ours (3D discovery)	×	×	intrinsics extrinsics	real

Table 4.4: Comparison of our work with representative related work for 3D pose using multi-view training. Previous works require either 3D or 2D supervision, or simulated environments to train jointly with reinforcement learning. Our method addresses a gap in discovering 3D keypoints from real videos without 2D or 3D supervision.

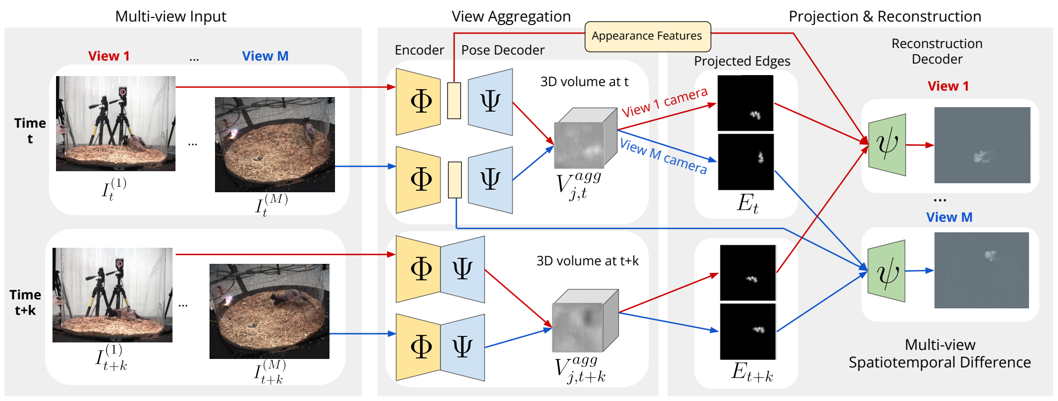


Figure 4.8: BKinD-3D: 3D keypoint discovery using 3D volume bottleneck. We start from input multi-view videos with known camera parameters, then unproject feature maps from geometric encoders into 3D volumes for timestamps t and $t + k$. We next aggregate 3D points from volumes into a single edge map at each timestamp, and use edges as input to the decoder alongside appearance features at time t. The model is trained using multi-view spatiotemporal difference reconstruction.

Method

Our method (BKinD-3D, Figure 4.8) uses multi-view spatiotemporal reconstruction to train an encoder-decoder architecture with 2D information aggregated to a 3D volumetric heatmap. Projections from the 3D heatmap in the form of agent skeletons are then used to reconstruct movement, represented by spatiotemporal difference, in each view.

Problem Setup. Given behavioral videos captured from M synchronized camera views, with known camera projection matrix $P^{(i)}$ for each camera $i \in \{1...M\}$, we aim to discover a set of J 3D keypoints $U_t \in \mathbb{R}^{J\times 3}$ on a single behaving agent, at each timestamp t. We assume access to camera projection matrices so that our model discovers 3D keypoints in the global coordinate frame.

During training, our model uses two timestamps in the video t and $t+k$ to compute the spatiotemporal difference in each view as the reconstruction target. In other words, for each camera view i, our training starts with a frame $I_t^{(i)}$ and a future frame $I_{t+k}^{(i)}$. During inference, only a single timestamp is required: once the model is trained, the model only needs $I_t^{(i)}$ for each camera view i.

In our model setup, the appearance encoder Φ, geometry decoder Ψ, and reconstruction decoder ψ are shared across views and timestamps (in our previous section [62], these networks are shared across timestamps, but only a single view is addressed). The appearance encoder Φ is used to generate appearance features, which are decoded into 2D heatmaps by the geometry decoder Ψ. These 2D heatmaps are then aggregated across views to form a 3D volumetric bottleneck (Section 4.5), which is processed by a volume-to-volume network ρ. We compute the 3D keypoints using spatial softmax on the 3D volume. Then, we project these keypoints to 2D, compute edges between points, and output these edges into the reconstruction decoder ψ (Section 4.5) for training. The reconstruction decoder ψ is only used during training, and not required for inference.

Feature Encoding. To start, we first compute appearance features from frame pairs $I_t^{(i)}$ and $I_{t+k}^{(i)}$ using the appearance encoder Φ: $\Phi(I_t^{(i)})$ and $\Phi(I_{t+k}^{(i)})$. These appearance features are then fed into the geometry decoder Ψ to generate 2D heatmaps $\Psi(\Phi(I_t^{(i)})) = H_t^{(i)}$ and $H_{t+k}^{(i)}$. Each 2D heatmap has C channels, where $H_{t,c}^{(i)}$ represents channel c of $H_t^{(i)}$.

View Aggregation using Volumetric Model. To aggregate information across views, we unproject our 2D heatmaps to a 3D volumetric bottleneck. We perform view aggregation separately across timestamps t and $t+k$.

We aggregate 2D heatmaps into a 3D volume similar to [26], which used previously for supervised 3D human pose estimation. One important difference is that in the supervised setting, an $L\times L\times L$ sized volume is drawn around the human pelvis, with L being around twice the size of a person. As we perform keypoint discovery, we do not have information on the location or size of the agent. Instead, we initialize our volume with L representing the maximum size of the space/room for the behaving agent.

This process aggregates 2D heatmaps $H_{t,c}^{(i)}$ for cameras $i \in \{1...M\}$ and channels $c \in \{1...C\}$ to 3D keypoints U_t, for timestamp t. Our volume is first discretized into voxels $V_{coords} \in \mathbb{R}^{B\times B\times B\times 3}$, where B represents the number of distinct coordinates in each dimension. Each voxel corresponds to a global 3D coordinate. These 3D coordinates are projected to a 2D plane using the projection matrices in each camera view i: $V_{proj}^{(i)} = P^{(i)}V_{coords}$. A volume $V_c^{(i)}$ is then created and filled for each camera view i and each channel c using bilinear sampling [27] from the corresponding 2D heatmap: $V_c^{(i)} = H_{t,c}^{(i)}\{V_{proj}^{(i)}\}$, where $\{\cdot\}$ denotes bilinear sampling.

We then aggregate these $V_c^{(i)}$ across views for each channel c using a softmax approach [26]:

$$V_c^{agg} = \sum_i \frac{\exp(V_c^{(i)})}{\sum_j \exp(V_c^{(j)})} \odot V_c^{(i)}.$$

V^{agg} is then mapped to 3D heatmaps corresponding to each joint using a volumetric convolutional network [41] ρ: $V^{agg*} = \rho(V^{agg})$. We compute the 3D spatial softmax over the volume, for each channel j of V_j^{agg*}, $j \in \{1...J\}$, to obtain the 3D keypoint locations U_t for timestamp t, as in [26]. In many supervised works, the keypoint locations U_t are optimized to match to ground truth 3D poses; however, we aim to discover 3D keypoints, and train our network by using U_t to decode spatiotemporal difference across views.

Projection and Reconstruction. In this step, we project the discovered 3D keypoints to a 2D representation in each view using camera parameters. For training, 2D representations in timestamps t and $t + k$ are used as input to the reconstruction decoder ψ. We train the 3D keypoints U_t at each timestamp t using multi-view spatiotemporal difference reconstruction. The target spatiotemporal difference is computed using the 2D image pair $I_t^{(i)}$ and $I_{t+k}^{(i)}$ at each view i.

First, we project the 3D keypoints using camera projection matrices into 2D keypoints $u_t^{(i)} = P^{(i)}U_t$. We create an edge representation for each view for each timestamp, which enables us to discover connections between points and enforce 3D joint length constraints. For each keypoint pair $u_{t,m}^{(i)}$ and $u_{t,n}^{(i)}$, we draw a differentiable edge map as a Gaussian along the line connecting them, similar to [22]:

$$E_{t,(m,n)}^{(i)}(\mathbf{p}) = \exp(d_{m,n}^{(i)}(\mathbf{p})^2/\sigma^2),$$

where σ controls the line thickness and $d_{m,n}(\mathbf{p})^{(i)}$ is the distance between pixel $\mathbf{p}$ and the line connecting $u_{t,m}^{(i)}$ and $u_{t,n}^{(i)}$. We then aggregate the edge heatmaps at each timestamp using a set of learned weights $w_{m,n}$ for each edge, where $w_{m,n}$ is shared across all timestamps and all views. An edge is active and connects two points if

$w_{m,n} > 0$, otherwise the points are not connected. Finally, we aggregate all edge heatmaps using the max across all edge pairs [22]:

$$E_t^{(i)}(\mathbf{p}) = \max_{m,n} w_{m,n} E_{t,(m,n)^{(i)}}(\mathbf{p}).$$

In our framework, for each view i, the decoder ψ uses the edge maps $E_t^{(i)}$ and $E_{t+k}^{(i)}$ as well as the appearance feature $\Phi(I_t^{(i)})$ for reconstructing the spatiotemporal difference across each view. The ground truth spatiotemporal difference is computed from the original images $S(I_t^{(i)}, I_{t+k}^{(i)})$. The reconstruction from the model is $\hat{S} = \psi(E_t^{(i)}, E_{t+k}^{(i)}, \Phi(I_t^{(i)}))$, through the 3D volumetric bottleneck in order to discover informative 3D keypoints for reconstructing agent movement.

Learning Formulation

The entire training pipeline (Figure 4.8) is differentiable, and we train the model end-to-end. We note that our model is only given multi-view video and corresponding camera parameters, without keypoint or bounding box supervision.

Multi-View Reconstruction Loss. Our multi-view spatiotemporal difference reconstruction is based on the single-view spatiotemporal difference described in the previous section on 2D keypoint discovery. We use SSIM as a reconstruction target and we compute a similarity map using local SSIM on corresponding patches between $I_t^{(i)}$ and $I_{t+k}^{(i)}$. This similarity map is negated to obtain the dissimilarity map used as the target: $S(I_t^{(i)}, I_{t+k}^{(i)})$.

We use perceptual loss [31] in each view between the target S and the reconstruction $\hat{S}$. This loss computes the L2 distance between features of the target and reconstruction computed from the VGG network ϕ [56]:

$$\mathcal{L}_{recon}^{(i)} = \left\| \phi(S(I_t^{(i)}, I_{t+T}^{(i)})) - \phi(\hat{S}(I_t^{(i)}, I_{t+T}^{(i)})) \right\|_2 . \tag{4.6}$$

The error is computed by comparing features from intermediate convolutional blocks of the network. Our final perceptual loss is summed over each view $\mathcal{L}_{recon} = \sum_i \mathcal{L}_{recon}^{(i)}$.

Learned Length Constraint. Since many animals have a rigid skeletal structure, we encourage that the length of active edges ($w_{m,n} > 0$ for point pairs m and n) are consistent across samples. We do not assume that these lengths and connections are known, such as previous work [65]; rather, they are learned during training. We do this by maintaining a running average of the length of all active edges $l_{avg(m,n)}$, and minimizing the difference between the average length and each sample $l_{m,n}$:

$$\mathcal{L}_{length} = \sum_m \sum_n \mathbb{1}_{w_{m,n}>0} \left\| l_{avg(m,n)} - l_{m,n} \right\|_2 . \tag{4.7}$$

During training, we update $l_{avg(m,n)}$ using an exponential running average and $w_{m,n}$ indicating edge weights for every pair is learned. Both of these parameters are shared across all viewpoints and timestamps. Notably, the length constraint is only applied to active edges, since there are many point pairs without rigid connections (e.g., elbow to feet), while we want to enforce this constraint only for rigid connections (e.g., elbow to wrist).

Separation Loss. To encourage unique keypoints to be discovered, we apply separation loss to our 3D keypoints, which has been previously studied in 2D [71, 62]. On a set of 3D keypoints U_{it}, where i is the index of a keypoint and t is the time, the separation loss is:

$$\mathcal{L}_s = \sum_{i \neq j} \exp\left(\frac{-(U_{it} - U_{jt})^2}{2\sigma_s^2}\right), \tag{4.8}$$

where σ_s is a hyperparameter that controls the strength of separation.

Training Objective. Our full training objective is the sum of the multi-view spatiotemporal reconstruction loss $\mathcal{L}_{recon}$, learned length constraints $\mathcal{L}_{length}$, and separation loss $\mathcal{L}_s$:

$$\mathcal{L} = \mathcal{L}_{recon} + \mathbb{1}_{epoch>e}(\omega_r \mathcal{L}_{length} + \omega_s \mathcal{L}_s). \tag{4.9}$$

Our model is trained using curriculum learning [2]. We only apply $\mathcal{L}_{length}$ and $\mathcal{L}_s$ when the keypoints are more consistent, after e epochs of training using reconstruction loss.

4.6 Experiments for 3D

We demonstrate BKinD-3D using real-world behavioral videos, using a human dataset and a recently released large-scale rat dataset (Section 7.4). We evaluate our discovered keypoints using a standard linear regression protocol based on previous works for 2D keypoint discovery [29, 62] (also described in Section 4.6).

Datasets

We demonstrate our method by evaluating it on two representative datasets: Human 3.6M and Rat7M. The datasets have different environments and focus on subjects of different sizes, with humans being about 1700mm tall and rats about 250mm long.

Human3.6M. We evaluate our method on Human3.6M to compare to recent works in self-supervised 3D from 2D [65]. Human3.6M [24] is a large-scale motion capture dataset with videos from 4 viewpoints. We follow the standard evaluation protocol [26, 34] to use subjects 1, 5, 6, 7, and 8 for training and 9 and 11 for testing. Our test set matches the set specified in [65] using every 16th frame (8516 test frame sets). Notably, unlike baselines such as [26], our method does not require any pre-processing with 2D bounding box annotations but rather is directly applied to the full image frame.

Rat7M. We also evaluate our method on Rat7M [15], a 3D pose dataset of rats moving in a behavioral arena. This dataset most closely matches the expected use case for our method, which is a dataset of non-human animal behavior in a static environment. Rat7M consists videos from 6 viewpoints captured at 1328×1048 resolution and 120Hz, along with ground truth annotations obtained from marker-based tracking. We train on subjects 1, 2, 3, 4, and test on subject 5, as in [15]. We train and evaluate on every 240th frame of each video (3083 train, 1934 test frame sets).

Model Comparisons

We compare our method with three main categories of baselines: supervised 3D pose estimation methods (for example, [26]), 3D pose estimation methods from 2D supervision (for example, [65]), and a 3D keypoint discovery method developed for control in simulation [7]. A more detailed comparison of methods in this space is in Table 4.4. For baselines with model variations, we use evaluation results from the version that is the closest to our model (multi-view inference, and camera parameters during inference). We note that all previous methods require additional 3D or 2D supervision, or jointly training a reinforcement learning policy in simulation [7], which we do not require for 3D keypoint discovery in real videos. Another notable difference is that previous methods typically pre-process video frames using detected or ground truth 2D bounding boxes [26], while our method does not require this pre-processing step.

Since 3D keypoint discovery has not been thoroughly explored, we additionally study methods in this area using multi-view 2D discovery and triangulation (Triang.+Reproj.), and multi-view 2D discovery with a depth map estimates (Depth Map), in addition to our volumetric approach (Section 4.5, BKinD-3D). For multi-view 2D discovery and triangulation, we use BKinD [62] (the work from our previous section) to discover 2D keypoints in each view, and perform triangulation using camera parameters to obtain 3D keypoints. We then project the 3D keypoints for multi-view reconstruction. We add an additional loss on the reprojection error to learn keypoints consistent across multiple views. For the depth map approach, in each camera view, we estimate 2D heatmaps corresponding to each keypoint alongside a view-specific depthmap estimate. The final 3D keypoints are then computed from a confidence-weighted average of each view's estimated 3D keypoint coordinates (from the per-view 2D heatmaps and depth estimates). More details on each method are in the supplementary materials of [61].

Training and Evaluation Procedure

We train our volumetric approach using the full objective (Eq 4.9). We scale images to 256×256 for training, with a frame gap of 0.4s for Human3.6M and 0.66s for Rat7M. We use a maximum volume size of 7500mm for Human3.6M and 1000mm for Rat7M. The results are computed for all 3D keypoint discovery methods with 15 keypoints unless otherwise specified. We train using videos from the train split with camera parameters provided by each dataset.

We evaluate our 3D keypoint discovery through keypoint regression based on similar methods from 2D, using a linear regressor without a bias term [62, 29, 71]. For this regression step, we extract our discovered 3D keypoints from a frozen network, and learn a linear regressor to map our discovered keypoints to the provided 3D keypoints in each of the training sets. We then perform evaluation on regressed keypoints on the test set.

For metrics, we compute Mean Per Joint Position Error (MPJPE) in line with previous works in 3D pose estimation [26, 25], which is the L2 distance between the regressed and ground truth 3D poses, accounting for the mean shift between the regressed and ground truth points. To compare to methods that require addition alignment before MPJPE computation (e.g., [65] which does not use camera parameters during inference), we also compute Procrustes aligned MPJPE (PMPJPE) [65, 34, 25]. PMPJPE applies the optimal rigid alignment to the predicted and ground truth 3D poses before metric computation.

Results

We evaluate our discovered keypoints quantitatively using keypoint regression on Human3.6M (Table 4.5) and Rat7M (Table 4.6). Over both datasets with diverse organisms, our approach generally outperforms all other fully self-supervised 3D keypoint discovery approaches. Additionally, among all the approaches we developed for 3D keypoint discovery, BKinD-3D using the volumetric bottleneck performs the best overall. Results demonstrate that BKinD-3D is directly applicable to discover 3D keypoints on novel model organisms, potentially very different in appearance or size, without 2D or 3D supervision.

Method	Supervision	PMPJPE ↓	MPJPE ↓
Supervised 3D			
Anipose [33]	2D only	-	33
Rhodin et al. [48]	3D/2D	52	67
Isakov et al. [26]	3D/2D	-	21
Supervised 2D + self-supervised 3D			
CanonPose [66]	2D	53	74
EpipolarPose [34]	2D	67	77
Iqbal et al. [25]	2D	55	69
MetaPose [65]	2D	32	-
3D Discovery + Regression			
Keypoint3D [7]	×	168	368
Ours:			
Triang+reproj	×	134	241
Depth Map	×	122	161
BKinD-3D	×	105	125

Table 4.5: Comparing performance with related work on Human3.6M. We note that previous approaches typically require additional 2D or 3D supervision, whereas our model discovers 3D keypoints directly from multi-view video. The 3D keypoint discovery models are evaluated using a linear regression protocol (Section 4.6).

Notably, on Human3.6M, Keypoint3D [7], developed for control of simulated videos, does not work well in our setting with real videos, and qualitative results demonstrate that this method was not able to discover keypoints that tracked the agent (supplementary materials of [61]).

Qualitative results. We find that the discovered points and skeletons are reasonable and look similar to the ground truth annotations for Human3.6M (Figure 4.9) and Rat7M (Figure 4.10). Furthermore, we find that a volumetric model with 30 keypoints learns a more detailed human skeleton representation than a model with 15 keypoints. For example, the model with 30 keypoints is able to track both legs, while the 15 keypoint model only tracks 1 leg; however, both models miss the knees. Importantly, our model discovers the skeleton in global coordinates, and is able to track the agent as they move around the space. More examples are in supplementary materials of [61].

While there exists a gap in terms of quantitative metrics between supervised methods and self-supervised 3D keypoint discovery, supervised methods require users to invest time and resources for annotations. In comparison, our method can be deployed out-of-the-box on new datasets and experiments with multi-view cameras.

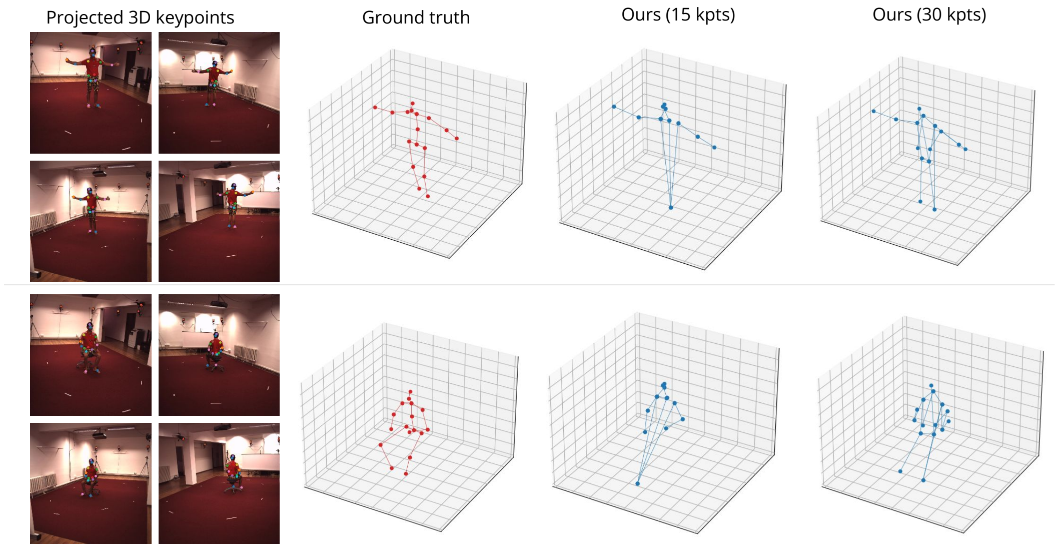


Figure 4.9: Qualitative results for 3D keypoint discovery on Human3.6M. Representative samples of 3D keypoints discovered from BKinD-3D without regression or alignment for 15 and 30 total discovered keypoints. We visualize all keypoints that are connected using the learned edge weights, and the projected 3D keypoints in the leftmost column are from the keypoint model with 30 discovered keypoints.

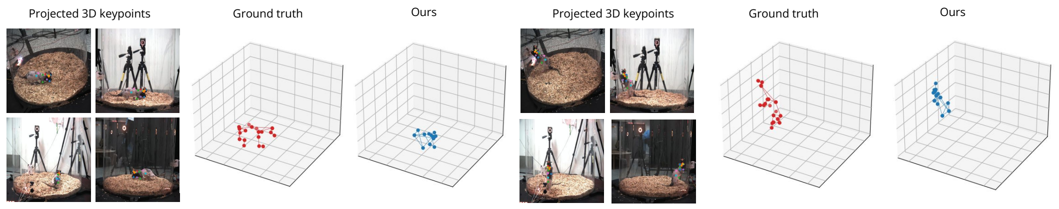


Figure 4.10: Qualitative results for 3D keypoint discovery on Rat7M. Representative samples of 3D keypoints discovered from BKinD-3D without regression or alignment. We visualize all connected keypoints using the learned edge weights and visualize the first 4 cameras (out of 6 cameras) in Rat7M for projected 3D keypoints.

Method	Supervision	PMPJPE ↓	MPJPE ↓
	Supervised 3D		
DANNCE [15]	3D	11	-
	3D Discovery + Regression		
Ours:			
Triang+reproj	×	21	108
Depth Map	×	27	56
BKinD-3D	×	24	76

Table 4.6: Comparison with 3D keypoint discovery methods on Rat7M. Results from the top three 3D keypoint discovery methods on Rat7M. The 3D keypoint discovery models are evaluated using a linear regression protocol (Section 4.6).

Our approach has closed the gap substantially to supervised methods compared to previous work, without requiring time-consuming 2D or 3D annotations. Qualitative results demonstrate that our approach is able to discover structure across diverse model organisms, providing a method for accelerating the study of organism movements in 3D.

Downstream Analysis. To further evaluate our keypoint discovery method, we use BKinD-3D keypoints as input to a 1D convolutional neural network (previously used in [58]) to predict action labels on Human3.6M. Notably, we found that our keypoints performs similarly to ground truth 3D points for action recognition, where Top 5 accuracy is 64.8% (GT), 61.0% (15 kpts), and 64.9% (30 kpts) (supplementary material of [61]).

References

[1] David J. Anderson and Pietro Perona. "Toward a Science of Computational Ethology." In: *Neuron* 84.1 (2014), pp. 18–31.

[2] Yoshua Bengio et al. "Curriculum Learning." In: *International Conference on Machine Learning (ICML)*. 2009.

[3] Kristin Branson et al. "High-Throughput Ethomics in Large Groups of Drosophila." In: *Nature Methods* 6.6 (2009), pp. 451–457.

[4] Xavier P. Burgos-Artizzu et al. "Social Behavior Recognition in Continuous Video." In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 1322–1329.

[5] Jennifer Cardona, Michael Howland, and John Dabiri. "Seeing the Wind: Visual Wind Speed Prediction with a Coupled Convolutional and Recurrent Neural Network." In: *Advances in Neural Information Processing Systems*. Ed. by Hanna Wallach et al. Vol. 32. Curran Associates, Inc., 2019.

[6] Jennifer L. Cardona and John O. Dabiri. "Wind Speed Inference from Environmental Flow-Structure Interactions, Part 2: Leveraging Unsteady Kinematics." In: *arXiv preprint arXiv:2107.09784* (2021).

[7] Boyuan Chen, Pieter Abbeel, and Deepak Pathak. "Unsupervised Learning of Visual 3D Keypoints for Control." In: *International Conference on Machine Learning*. PMLR. 2021, pp. 1539–1549.

[8] Ching-Hang Chen and Deva Ramanan. "3D Human Pose Estimation = 2D Pose Estimation + Matching." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7035–7043.

[9] Ching-Hang Chen et al. "Unsupervised 3D Pose Estimation with Geometric Self-Supervision." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5714–5724.

[10] Long Chen et al. “Cross-View Tracking for Multi-Human 3D Pose Estimation at Over 100 FPS.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 3279–3288.

[11] Yilun Chen et al. “Cascaded Pyramid Network for Multi-Person Pose Estimation.” In: *Computing Research Repository* abs/1711.07319 (2017).

[12] Bowen Cheng et al. “HigherHRNet: Scale-Aware Representation Learning for Bottom-Up Human Pose Estimation.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[13] John H. Costello et al. “The Hydrodynamics of Jellyfish Swimming.” In: *Annual Review of Marine Science* 13 (2021), pp. 375–396.

[14] Heiko Dankert et al. “Automated Monitoring and Analysis of Social Behavior in Drosophila.” In: *Nature Methods* 6.4 (2009), pp. 297–303.

[15] Timothy W. Dunn et al. “Geometric Deep Learning Enables 3D Kinematic Profiling Across Species and Environments.” In: *Nature Methods* 18.5 (2021), pp. 564–573.

[16] S. E. Roian Egnor and Kristin Branson. “Computational Analysis of Behavior.” In: *Annual Review of Neuroscience* 39 (2016), pp. 217–236.

[17] Eyrun Eyjolfsdottir et al. “Detecting Social Actions of Fruit Flies.” In: *European Conference on Computer Vision*. Springer. 2014, pp. 772–787.

[18] Eyrun Eyjolfsdottir et al. “Learning Recurrent Representations for Hierarchical Behavior Modeling.” In: *International Conference on Learning Representations* (2017).

[19] Adam Gosztolai et al. “LiftPose3D: A Deep Learning-Based Approach for Transforming Two-Dimensional to Three-Dimensional Poses in Laboratory Animals.” In: *Nature Methods* 18.8 (2021), pp. 975–981.

[20] Jacob M. Graving et al. “DeepPoseKit, a Software Toolkit for Fast and Robust Animal Pose Estimation Using Deep Learning.” In: *eLife* 8 (2019), e47994.

[21] Semih Günel et al. “DeepFly3D: A Deep Learning-Based Approach for 3D Limb and Appendage Tracking in Tethered, Adult Drosophila.” In: *eLife* 8 (Oct. 2019). Ed. by Timothy O’Leary, Ronald L Calabrese, and Josh W Shaevitz, e48571. ISSN: 2050-084X. DOI: `10.7554/eLife.48571`.

[22] Xingzhe He, Bastian Wandt, and Helge Rhodin. “AutoLink: Self-Supervised Learning of Human Skeletons and Object Outlines by Linking Keypoints.” In: *arXiv preprint arXiv:2205.10636* (2022).

[23] Weizhe Hong et al. “Automated Measurement of Mouse Social Behaviors Using Depth Sensing, Video Tracking, and Machine Learning.” In: *Proceedings of the National Academy of Sciences* 112.38 (2015), E5351–E5360.

[24] Catalin Ionescu et al. “Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.7 (2013), pp. 1325–1339.

[25] Umar Iqbal, Pavlo Molchanov, and Jan Kautz. “Weakly-Supervised 3D Human Pose Learning via Multi-View Images in the Wild.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5243–5252.

[26] Karim Iskakov et al. “Learnable Triangulation of Human Pose.” In: *The IEEE International Conference on Computer Vision (ICCV)*. 2019.

[27] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. “Spatial Transformer Networks.” In: *Advances in Neural Information Processing Systems* 28 (2015).

[28] Tomas Jakab et al. “Self-Supervised Learning of Interpretable Keypoints from Unlabelled Videos.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8787–897.

[29] Tomas Jakab et al. “Unsupervised Learning of Object Landmarks Through Conditional Image Generation.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[30] Hueihan Jhuang et al. “Automated Home-Cage Behavioural Phenotyping of Mice.” In: *Nature Communications* 1.1 (2010), pp. 1–10.

[31] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. “Perceptual Losses for Real-Time Style Transfer and Super-Resolution.” In: *European Conference on Computer Vision*. 2016.

[32] Mayank Kabra et al. “JAABA: Interactive Machine Learning for Automatic Annotation of Animal Behavior.” In: *Nature Methods* 10.1 (2013), p. 64.

[33] Pierre Karashchuk et al. “Anipose: A Toolkit for Robust Markerless 3D Pose Estimation.” In: *Cell Reports* 36.13 (2021), p. 109730.

[34] Muhammed. Kocabas, Salih Karagoz, and Emre Akbas. “Self-Supervised Learning of 3D Human Pose Using Multi-View Geometry.” In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[35] Jingyuan Liu et al. “Normalized Human Pose Features for Human Action Video Alignment.” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 11521–11531.

[36] Dominik Lorenz et al. “Unsupervised Part-Based Disentangling of Object Shape and Appearance.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[37] Jesse D. Marshall et al. “Leaving Flatland: Advances in 3D Behavioral Measurement.” In: *Current Opinion in Neurobiology* 73 (2022), p. 102522.

[38] Julian Marstaller, Frederic Tausch, and Simon Stock. “DeepBees: Building and Scaling Convolutional Neuronal Nets for Fast and Large-Scale Visual Monitoring of Bee Hives.” In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2019, pp. 0-0.

[39] Julieta Martinez et al. “A Simple Yet Effective Baseline for 3D Human Pose Estimation.” In: *International Conference on Computer Vision*. 2017.

[40] Alexander Mathis et al. “DeepLabCut: Markerless Pose Estimation of User-Defined Body Parts with Deep Learning.” In: *Nature Neuroscience* (2018). URL: `https://www.nature.com/articles/s41593-018-0209-y`.

[41] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. “V2V-PoseNet: Voxel-to-Voxel Prediction Network for Accurate 3D Hand and Human Pose Estimation from a Single Depth Map.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5079–5088.

[42] Alejandro Newell, Kaiyu Yang, and Jia Deng. “Stacked Hourglass Networks for Human Pose Estimation.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016.

[43] Simon R.O. Nilsson et al. “Simple Behavioral Analysis (SimBA)–an Open Source Toolkit for Computer Classification of Complex Social Behaviors in Experimental Animals.” In: *BioRxiv* (2020).

[44] Dario Pavllo et al. “3D Human Pose Estimation in Video with Temporal Convolutions and Semi-Supervised Training.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7753–7762.

[45] Talmo D. Pereira, Joshua W. Shaevitz, and Mala Murthy. “Quantifying Behavior to Understand the Brain.” In: *Nature Neuroscience* 23.12 (2020), pp. 1537–1549.

[46] Talmo D. Pereira et al. “SLEAP: Multi-Animal Pose Tracking.” In: *BioRxiv* (2020).

[47] Mir Imtiaz Hossain Rayat and James J. Little. “Exploiting Temporal Information for 3D Human Pose Estimation.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 68–84.

[48] Helge Rhodin et al. “Learning Monocular 3D Human Pose Estimation from Multi-View Images.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8437–8446.

[49] Serim Ryou, Seong-Gyun Jeong, and Pietro Perona. “Anchor Loss: Modulating Loss Scale Based on Prediction Difficulty.” In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2019.

[50] Serim Ryou and Pietro Perona. “Weakly Supervised Keypoint Discovery.” In: *Computing Research Repository* abs/2109.13423 (2021). URL: `https://arxiv.org/abs/2109.13423`.

[51] Nikolaos Sarafianos et al. “3D Human Pose Estimation: A Review of the Literature and Analysis of Covariates.” In: *Computer Vision and Image Understanding* 152 (2016), pp. 1–20.

[52] Luca Schmidtke et al. “Unsupervised Human Pose Estimation Through Transforming Shape Templates.” In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 2484–2494.

[53] Cristina Segalin, Jalani Williams, Tomomi Karigo, May Hui, Moriel Zelikowsky, Jennifer J. Sun, Pietro Perona, David J. Anderson, and Ann Kennedy. “The Mouse Action Recognition System (MARS) Software Pipeline for Automated Analysis of Social Behaviors in Mice.” In: *eLife* 10 (2021), e63720.

[54] Evan Shelhamer, Jonathan Long, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 640–651.

[55] Brian M. de Silva et al. “Discovery of Physics from Data: Universal Laws and Discrepancies.” In: *Frontiers in Artificial Intelligence* 3 (2020), p. 25.

[56] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” In: *Computing Research Repository* abs/1409.1556 (2014).

[57] Jennifer J. Sun, Ann Kennedy, Eric Zhan, David J. Anderson, Yisong Yue, and Pietro Perona. “Task Programming: Learning Data Efficient Behavior Representations.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 2876–2885. URL: `https://arxiv.org/pdf/2011.13917.pdf`.

[58] Jennifer J. Sun, Tomomi Karigo, Dipam Chakraborty, Sharada Mohanty, Benjamin Wild, Quan Sun, Chen Chen, David Anderson, Pietro Perona, et al. “The Multi-Agent Behavior Dataset: Mouse Dyadic Social Interactions.” In: *Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track* (2021).

[59] Jennifer J. Sun, Jiaping Zhao, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, and Ting Liu. “View-Invariant Probabilistic Embedding for Human Pose.” In: *European Conference on Computer Vision (ECCV)* (2020), pp. 53–70. URL: `https://arxiv.org/pdf/1912.01001.pdf`.

[60] Xiao Sun et al. "Integral Human Pose Regression." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 529–545.

[61] Jennifer J. Sun*, Lili Karashchuk*, Amil Dravid*, Serim Ryou, Sonia Fereidooni, John C Tuthill, Aggelos Katsaggelos, Bingni W Brunton, Georgia Gkioxari, Ann Kennedy, et al. "BKinD-3D: Self-Supervised 3D Keypoint Discovery from Multi-View Videos." In: (2023), pp. 9001–9010. URL: `https://arxiv.org/pdf/2212.07401.pdf`.

[62] Jennifer J. Sun*, Serim Ryou*, Roni H. Goldshmid, Brandon Weissbourd, John O. Dabiri, David J. Anderson, Ann Kennedy, Yisong Yue, and Pietro Perona. "Self-Supervised Keypoint Discovery in Behavioral Videos." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 2171–2180. URL: `https://arxiv.org/pdf/2112.05121.pdf`.

[63] Wei Tang, Pei Yu, and Ying Wu. "Deeply Learned Compositional Models for Human Pose Estimation." In: *The European Conference on Computer Vision (ECCV)*. Sept. 2018.

[64] James Thewlis, Hakan Bilen, and Andrea Vedaldi. "Unsupervised Learning of Object Landmarks by Factorized Spatial Embeddings." In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[65] Ben Usman et al. "MetaPose: Fast 3D Pose from Multiple Views without 3D Supervision." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 6759–6770.

[66] Bastian Wandt et al. "CanonPose: Self-Supervised Monocular 3D Human Pose Estimation in the Wild." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13294–13304.

[67] Jinbao Wang et al. "Deep 3D Human Pose Estimation: A Review." In: *Computer Vision and Image Understanding* 210 (2021), p. 103225.

[68] Jingdong Wang et al. "Deep High-Resolution Representation Learning for Visual Recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43 (2021), pp. 3349–3364.

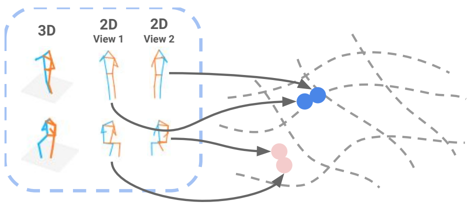
[69] Zhou Wang et al. "Image Quality Assessment: From Error Visibility to Structural Similarity." In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612.

[70] Shih-En Wei et al. "Convolutional Pose Machines." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

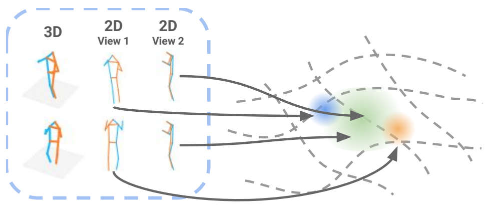
[71] Yuting Zhang et al. "Unsupervised Discovery of Object Landmarks as Structural Representations." In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2018. DOI: `10.1109/CVPR.2018.00285`.

Chapter 5

POSE REPRESENTATIONS



(a) View-Invariant Pose Embeddings (VIPE).



(b) Probabilistic View-Invariant Pose Embeddings (Pr-VIPE).

Figure 5.1: We embed 2D poses such that our embeddings are (a) view-invariant (2D projections of similar 3D poses are embedded close) and (b) probabilistic (embeddings are distributions that cover different 3D poses projecting to the same input 2D pose).

This chapter is mainly based on the following papers:

[1] Ting Liu*, Jennifer J. Sun*, Long Zhao, Jiaping Zhao, Liangzhe Yuan, Yuxiao Wang, Liang-Chieh Chen, Florian Schroff, and Hartwig Adam. "View-Invariant, Occlusion-Robust Probabilistic Embedding for Human Pose." In: *International Journal of Computer Vision* 130.1 (2022), pp. 111–135. URL: `https://arxiv.org/pdf/2010.13321.pdf`.

[2] Jennifer J. Sun, Jiaping Zhao, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, and Ting Liu. "View-Invariant Probabilistic Embedding for Human Pose." In: *European Conference on Computer Vision (ECCV)* (2020), pp. 53–70. URL: `https://arxiv.org/pdf/1912.01001.pdf`.

Abstract. Depictions of similar human body configurations can vary with changing viewpoints. Using only 2D information, we would like to enable vision algorithms to recognize similarity in human body poses across multiple views. This ability is useful for analyzing body movements and human behaviors in images and videos. In this paper, we propose an approach for learning a compact view-invariant embedding space from 2D joint keypoints alone, without explicitly predicting 3D poses. Since 2D poses are projected from 3D space, they have an inherent ambiguity, which is difficult to represent through a deterministic mapping. Hence, we use probabilistic

embeddings to model this input uncertainty. Experimental results show that our embedding model achieves higher accuracy when retrieving similar poses across different camera views, in comparison with 2D-to-3D pose lifting models. We also demonstrate the effectiveness of applying our embeddings to view-invariant action recognition and video alignment. Our code is available at `https://github.com/google-research/google-research/tree/master/poem`.

5.1 Introduction

When we represent three-dimensional (3D) human bodies in two dimensions (2D), the same human pose can appear different across camera views. There can be significant visual variations from a change in viewpoint due to changing relative depth of body parts and self-occlusions. Despite these variations, humans have the ability to recognize similar 3D human body poses in images and videos. This ability is useful for computer vision tasks where changing viewpoints should not change the labels of the task. We explore how we can embed 2D visual information of human poses to be consistent across camera views. We show that these embeddings are useful for tasks such as view-invariant pose retrieval, action recognition, and video alignment.

Inspired by 2D-to-3D lifting models [33], we learn view invariant embeddings directly from 2D pose keypoints. As illustrated in Fig. 5.1, we explore whether view invariance of human bodies can be achieved from 2D poses alone, without predicting 3D pose. Typically, embedding models are trained from images using deep metric learning techniques [36, 14, 8]. However, images with similar human poses can appear different because of changing viewpoints, subjects, backgrounds, clothing, etc. As a result, it can be difficult to understand errors in the embedding space from a specific factor of variation. Furthermore, multi-view image datasets for human poses are difficult to capture in the wild with 3D groundtruth annotations. In contrast, our method leverages existing 2D keypoint detectors: using 2D keypoints as inputs allows the embedding model to focus on learning view invariance. Our 2D keypoint embeddings can be trained using datasets in lab environments, while having the model generalize to in-the-wild data. Additionally, we can easily augment training data by synthesizing multi-view 2D poses from 3D poses through perspective projection.

Another aspect we address is input uncertainty. The input to our embedding model is 2D human pose, which has an inherent ambiguity. Many valid 3D poses can project to the same or very similar 2D pose [1]. This input uncertainty is difficult to represent using deterministic mappings to the embedding space (point embeddings) [38, 24]. Our embedding space consists of probabilistic embeddings based on multivariate Gaussians, as shown in Fig. 5.1b. We show that the learned variance from our method correlates with input 2D ambiguities. We call our approach Pr-VIPE for **Pr**obabilistic **V**iew-**I**nvariant **P**ose **E**mbeddings. The non-probabilistic, point embedding formulation will be referred to as VIPE.

We show that our embedding is applicable to subsequent vision tasks such as pose retrieval [36, 21], video alignment [11], and action recognition [62, 18]. One direct application is pose-based image retrieval. Our embedding enables users to search images by fine-grained pose, such as jumping with hands up, riding bike with one hand waving, and many other actions that are potentially difficult to pre-define. The importance of this application is further highlighted by works such as [36, 21]. Compared with using 3D keypoints with alignment for retrieval, our embedding enables efficient similarity comparisons in Euclidean space.

Contributions. Our main contribution is the method for learning an embedding space where 2D pose embedding distances correspond to their similarities in absolute 3D pose space. We also develop a probabilistic formulation that captures 2D pose ambiguity. We use cross-view pose retrieval to evaluate the view-invariant property: given a monocular pose image, we retrieve the same pose from different views without using camera parameters. Our results suggest 2D poses are sufficient to achieve view invariance without image context, and we do not have to predict 3D pose coordinates to achieve this. We also demonstrate the use of our embeddings for action recognition and video alignment.

5.2 Related Work

Metric Learning

We are working to understand similarity in human poses across views. Most works that aim to capture similarity between inputs generally apply techniques from metric learning. Objectives such as contrastive loss (based on pair matching) [4, 12, 38] and triplet loss (based on tuple ranking) [58, 51, 59, 13] are often used to push together/pull apart similar/dissimilar examples in embedding space. The number of possible training tuples increases exponentially with respect to the number of samples in the tuple, and not all combinations are equally informative. To find informative training tuples, various mining strategies are proposed [51, 60, 39, 13].

In particular, semi-hard triplet mining has been widely used [51, 60, 43]. This mining method finds negative examples that are fairly hard as to be informative but not too hard for the model. The hardness of a negative sample is based on its embedding distance to the anchor. Commonly, this distance is the Euclidean distance [58, 59, 51, 13], but any differentiable distance function could be applied [13]. [16, 19] show that alternative distance metrics also work for image and object retrieval.

In our work, we learn a mapping from Euclidean embedding distance to a probabilistic similarity score. This probabilistic similarity captures closeness in 3D pose space from 2D poses. Our work is inspired by the mapping used in soft contrastive loss [38] for learning from an occluded N-digit MNIST dataset.

Most of the papers discussed above involve deterministically mapping inputs to point embeddings. There are works that also map inputs to probabilistic embeddings. Probabilistic embeddings have been used to model specificity of word embeddings [57], uncertainty in graph representations [3], and input uncertainty due to occlusion [38]. We will apply probabilistic embeddings to address inherent ambiguities in 2D pose due to 3D-to-2D projection.

Human Pose Estimation

3D human poses in a global coordinate frame are view-invariant, since images across views are mapped to the same 3D pose. However, as mentioned by [33], it is difficult to infer the 3D pose in an arbitrary global frame since any changes to the frame does not change the input data. Many approaches work with poses in the camera coordinate system [33, 6, 44, 47, 64, 54, 48, 55, 7], where the pose description changes based on viewpoint. While our work focuses on images with a single person, there are other works focusing on describing poses of multiple people [50].

Our approach is similar in setup to existing 3D lifting pose estimators [33, 6, 44, 47, 9] in terms of using 2D pose keypoints as input. The difference is that lifting models are trained to regress to 3D pose keypoints, while our model is trained using metric learning and outputs an embedding distribution. Some recent works also use multi-view datasets to predict 3D poses in the global coordinate frame [45, 26, 20, 49, 56]. Our work differs from these methods with our goal (view-invariant embeddings), task (cross-view pose retrieval), and approach (metric learning). Another work on pose retrieval [36] embeds images with similar 2D poses in the same view close together. Our method focuses on learning view invariance, and we also differ from [36] in method (probabilistic embeddings).

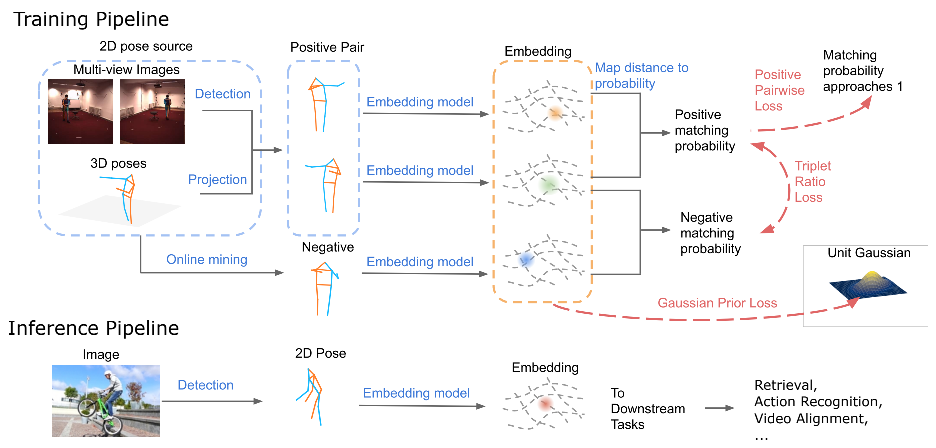


Figure 5.2: Overview of Pr-VIPE model training and inference. Our model takes keypoint input from a single 2D pose (detected from images and/or projected from 3D poses) and predicts embedding distributions. Three losses are applied during training.

View Invariance and Object Retrieval

When we capture a 3D scene in 2D as images or videos, changing the viewpoint often does not change other properties of the scene. The ability to recognize visual similarities across viewpoints is helpful for a variety of vision tasks, such as motion analysis [23, 22], tracking [40], vehicle and human re-identification [8, 63], object classification and retrieval [27, 15, 14], and action recognition [46, 29, 61, 28].

Some of these works focus on metric learning for object retrieval. Their learned embedding spaces place different views of the same object class close together. Our work on human pose retrieval differs in a few ways. Our labels are continuous 3D poses, whereas in object recognition tasks, each embedding is associated with a discrete class label. Furthermore, we embed 2D poses, while these works embed images. Our approach allows us to investigate the impact of input 2D uncertainty with probabilistic embeddings and explore confidence measures to cross-view pose retrieval. We hope that our work provides a novel perspective on view invariance for human poses.

5.3 View-Invariant Probabilistic Embeddings

The training and inference framework of Pr-VIPE is illustrated in Fig. 5.2. Our goal is to embed 2D poses such that distances in the embedding space correspond to similarities of their corresponding absolute 3D poses in Euclidean space. We achieve this view invariance property through our triplet ratio loss (Section 5.3), which pushes together/pull apart 2D poses corresponding to similar/dissimilar 3D poses. The positive pairwise loss (Section 5.3) is applied to increase the matching probability of similar poses. Finally, the Gaussian prior loss (Section 5.3) helps regularize embedding magnitude and variance.

Matching Definition

The 3D pose space is continuous, and two 3D poses can be trivially different without being identical. We define two 3D poses to be matching if they are visually similar regardless of viewpoint. Given two sets of 3D keypoints $(\boldsymbol{y}_i, \boldsymbol{y}_j)$, we define a matching indicator function

$$m_{ij} = \begin{cases} 1, & \text{if NP-MPJPE}(\boldsymbol{y}_i, \boldsymbol{y}_j) \leqslant \kappa \\ 0, & \text{otherwise,} \end{cases} \tag{5.1}$$

where κ controls visual similarity between matching poses. Here, we use mean per joint position error (MPJPE) [17] between the two sets of 3D pose keypoints as a proxy to quantify their visual similarity. Before computing MPJPE, we normalize the 3D poses and apply Procrustes alignment between them. The reason is that we want our model to be view-invariant and to disregard rotation, translation, or scale differences between 3D poses. We refer to this normalized, Procrustes aligned MPJPE as **NP-MPJPE**.

Triplet Ratio Loss

The triplet ratio loss aims to embed 2D poses based on the matching indicator function (5.1). Let n be the dimension of the input 2D pose keypoints $\boldsymbol{x}$, and d be the dimension of the output embedding. We would like to learn a mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}^d$, such that $D(\boldsymbol{z}_i, \boldsymbol{z}_j) < D(\boldsymbol{z}_i, \boldsymbol{z}_{j'}), \forall m_{ij} > m_{ij'}$, where $\boldsymbol{z} = f(\boldsymbol{x})$, and $D(\boldsymbol{z}_i, \boldsymbol{z}_j)$ is an embedding space distance measure.

For a pair of input 2D poses $(\boldsymbol{x}_i, \boldsymbol{x}_j)$, we define $p(m|\boldsymbol{x}_i, \boldsymbol{x}_j)$ to be the probability that their corresponding 3D poses $(\boldsymbol{y}_i, \boldsymbol{y}_j)$ match, that is, they are visually similar. While it is difficult to define this probability directly, we propose to assign its values by estimating $p(m|\boldsymbol{z}_i, \boldsymbol{z}_j)$ via metric learning. We know that if two 3D poses are identical, then $p(m|\boldsymbol{x}_i, \boldsymbol{x}_j) = 1$, and if two 3D poses are sufficiently different, $p(m|\boldsymbol{x}_i, \boldsymbol{x}_j)$ should be small. For any given input triplet $(\boldsymbol{x}_i, \boldsymbol{x}_{i^+}, \boldsymbol{x}_{i^-})$ with $m_{i,i^+} > m_{i,i^-}$, we want

$$\frac{p(m|\boldsymbol{z}_i, \boldsymbol{z}_{i^+})}{p(m|\boldsymbol{z}_i, \boldsymbol{z}_{i^-})} \geqslant \beta, \tag{5.2}$$

where $\beta > 1$ represents the ratio of the matching probability of a similar 3D pose pair to that of a dissimilar pair. Applying negative logarithm to both sides, we have

$$(-\log p(m|\boldsymbol{z}_i, \boldsymbol{z}_{i^+})) - (-\log p(m|\boldsymbol{z}_i, \boldsymbol{z}_{i^-})) \leqslant -\log \beta. \tag{5.3}$$

Notice that the model can be trained to satisfy this with the triplet loss framework [51]. Given batch size N, we define triplet ratio loss $\mathcal{L}_{\text{ratio}}$ as

$$\mathcal{L}_{\text{ratio}} = \sum_{i=1}^{N} \max(0, D_m(\boldsymbol{z}_i, \boldsymbol{z}_{i^+}) - D_m(\boldsymbol{z}_i, \boldsymbol{z}_{i^-}) + \alpha)), \tag{5.4}$$

with distance kernel $D_m(\boldsymbol{z}_i, \boldsymbol{z}_j) = -\log p(m|\boldsymbol{z}_i, \boldsymbol{z}_j)$ and margin $\alpha = \log \beta$. To form a triplet $(\boldsymbol{x}_i, \boldsymbol{x}_{i^+}, \boldsymbol{x}_{i^-})$, we set the anchor $\boldsymbol{x}_i$ and positive $\boldsymbol{x}_{i^+}$ to be projected from the same 3D pose and perform online semi-hard negative mining [51] to find $\boldsymbol{x}_{i^-}$.

It remains for us to compute matching probability using our embeddings. To compute $p(m|\boldsymbol{z}_i, \boldsymbol{z}_j)$, we use the formulation proposed by [38]:

$$p(m|\boldsymbol{z}_i, \boldsymbol{z}_j) = \sigma(-a||\boldsymbol{z}_i - \boldsymbol{z}_j||_2 + b), \tag{5.5}$$

where σ is a sigmoid function, and the trainable scalar parameters $a > 0$ and $b \in \mathbb{R}$ calibrate embedding distances to probabilistic similarity.

Positive Pairwise Loss

The positive pairs in our triplets have identical 3D poses. We would like them to have high matching probabilities, which can be encouraged by adding the positive pairwise loss

$$\mathcal{L}_{\text{positive}} = \sum_{i=1}^{N} -\log p(m|\boldsymbol{z}_i, \boldsymbol{z}_{i^+}). \tag{5.6}$$

The combination of $\mathcal{L}_{\text{ratio}}$ and $\mathcal{L}_{\text{positive}}$ can be applied to training point embedding models, which we refer to as VIPE in this paper.

Probabilistic Embeddings

In this section, we discuss the extension of VIPE to the probabilistic formulation Pr-VIPE. The inputs to our model, 2D pose keypoints, are inherently ambiguous, and there are many valid 3D poses projecting to similar 2D poses [1]. This input uncertainty can be difficult to model using point embeddings [24, 38]. We investigate representing this uncertainty using distributions in the embedding space by mapping 2D poses to probabilistic embeddings: $\boldsymbol{x} \rightarrow p(\boldsymbol{z}|\boldsymbol{x})$. Similar to [38], we extend the input matching probability (5.5) to using probabilistic embeddings as $p(m|\boldsymbol{x}_i, \boldsymbol{x}_j) = \int p(m|\boldsymbol{z}_i, \boldsymbol{z}_j)p(\boldsymbol{z}_i|\boldsymbol{x}_i)p(\boldsymbol{z}_j|\boldsymbol{x}_j)\mathrm{d}\boldsymbol{z}_i\mathrm{d}\boldsymbol{z}_j$, which can be approximated using Monte-Carlo sampling with K samples drawn from each distribution as

$$p(m|\boldsymbol{x}_i, \boldsymbol{x}_j) \approx \frac{1}{K^2} \sum_{k_1=1}^{K} \sum_{k_2=1}^{K} p(m|\boldsymbol{z}_i^{(k_1)}, \boldsymbol{z}_j^{(k_2)}). \tag{5.7}$$

We model $p(\boldsymbol{z}|\boldsymbol{x})$ as a d-dimensional Gaussian with a diagonal covariance matrix.

The model outputs mean $\mu(\boldsymbol{x}) \in \mathbb{R}^d$ and covariance $\Sigma(\boldsymbol{x}) \in \mathbb{R}^d$ with shared base network and different output layers. We use the reparameterization trick [25] during sampling.

In order to prevent variance from collapsing to zero and to regularize embedding mean magnitudes, we place a unit Gaussian prior on our embeddings with KL divergence by adding the Gaussian prior loss

$$\mathcal{L}_{\text{prior}} = \sum_{i=1}^{N} D_{\text{KL}}(\mathcal{N}(\mu(\boldsymbol{x}_i), \Sigma(\boldsymbol{x}_i)) \parallel \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})). \tag{5.8}$$

Inference. At inference time, our model takes a single 2D pose (either from detection or projection) and outputs the mean and the variance of the embedding Gaussian distribution.

Camera Augmentation

Our triplets can be made of detected and/or projected 2D keypoints as shown in Fig. 5.2. When we train only with detected 2D keypoints, we are constrained to the camera views in training images. To reduce overfitting to these camera views, we perform camera augmentation by generating triplets using detected keypoints alongside projected 2D keypoints at random views.

To form triplets using multi-view image pairs, we use detected 2D keypoints from different views as anchor-positive pairs. To use projected 2D keypoints, we perform two random rotations to a normalized input 3D pose to generate two 2D poses from different views for anchor/positive. Camera augmentation is then performed by using a mixture of detected and projected 2D keypoints. We find that training using camera augmentation can help our models learn to generalize better to unseen views (Section 5.4).

Implementation Details

We normalize 3D poses similar to [7], and we perform instance normalization to 2D poses. The backbone network architecture for our model is based on [33]. We use $d = 16$ as a good trade-off between embedding size and accuracy. To weigh different losses, we use $w_{\text{ratio}} = 1$, $w_{\text{positive}} = 0.005$, and $w_{\text{prior}} = 0.001$. We choose $\beta = 2$ for the triplet ratio loss margin and $K = 20$ for the number of samples. The matching NP-MPJPE threshold is $\kappa = 0.1$ for all training and evaluation. Our approach does not rely on a particular 2D keypoint detector, and we use PersonLab [41] for our experiments. For random rotation in camera augmentation, we uniformly sample azimuth angle between $\pm 180°$, elevation between $\pm 30°$, and roll between $\pm 30°$. Our implementation is in TensorFlow, and all the models are trained with CPUs. More details and ablation studies on hyperparamters are provided in the supplementary materials of [53].

5.4 Experiments

We demonstrate the performance of our model through pose retrieval across different camera views (Section 5.4). We further show our embeddings can be directly applied to downstream tasks, such as action recognition (Section 5.4) and video alignment (Section 5.4), without any additional training.

Datasets

For all the experiments in this paper, we only train on a subset of the Human3.6M [17] dataset. For pose retrieval experiments, we validate on the Human3.6M hold-out set and test on another dataset (MPI-INF-3DHP [34]), which is unseen during training and free from parameter tuning. We also present qualitative results on MPII Human Pose [2], for which 3D groundtruth is not available. Additionally, we directly use our embeddings for action recognition and sequence alignment on Penn Action [62].

Human3.6M (H3.6M) H3.6M is a large human pose dataset recorded from 4 chest level cameras with 3D pose groundtruth. We follow the standard protocol [33]: train on Subject 1, 5, 6, 7, and 8, and hold out Subject 9 and 11 for validation. For evaluation, we remove near-duplicate 3D poses within 0.02 NP-MPJPE, resulting in a total of 10910 evaluation frames per camera. This process is camera-consistent, meaning if a frame is selected under one camera, it is selected under all cameras, so that the perfect retrieval result is possible.

MPI-INF-3DHP (3DHP) 3DHP is a more recent human pose dataset that contains 14 diverse camera views and scenarios, covering more pose variations than H3.6M [34]. We use 11 cameras from this dataset and exclude the 3 cameras with overhead views. Similar to H3.6M, we remove near-duplicate 3D poses, resulting in 6824 frames per camera. We use all 8 subjects from the train split of 3DHP. **This dataset is only used for testing.**

MPII Human Pose (2DHP) This dataset is commonly used in 2D pose estimation, containing 25K images from YouTube videos. Since groundtruth 3D poses are not available, we show qualitative results on this dataset.

Penn Action This dataset contains 2326 trimmed videos for 15 pose-based actions from different views. We follow the standard protocol [37] for our action classification and video alignment experiments.

View-Invariant Pose Retrieval

Given multi-view human pose datasets, we query using detected 2D keypoints from one camera view and find the nearest neighbors in the embedding space from a different camera view. We iterate through all camera pairs in the dataset as query and index. Results averaged across all cameras pairs are reported.

Evaluation Procedure. We report Hit@k with $k = 1$, 10, and 20 on pose retrievals, which is the percentage of top-k retrieved poses that have at least one accurate retrieval. A retrieval is considered accurate if the 3D groundtruth from the retrieved pose satisfies the matching function (5.1) with $\kappa = 0.1$.

Baseline Approaches. We compare Pr-VIPE with 2D-to-3D lifting models [33] and $L2$-VIPE. $L2$-VIPE outputs $L2$-normalized point embeddings, and is trained with the squared $L2$ distance kernel, similar to [51].

For fair comparison, we use the same backbone network architecture for all the models. Notably, this architecture [33] has been tuned for lifting tasks on H3.6M. Since the estimated 3D poses in camera coordinates are not view-invariant, we apply normalization and Procrustes alignment to align the estimated 3D poses between index and query for retrieval. In comparison, our embeddings do not require any alignment or other post-processing during retrieval.

For Pr-VIPE, we retrieve poses using nearest neighbors in the embedding space with respect to the sampled matching probability (5.7), which we refer to as retrival confidence. We present the results on the VIPE models with and without camera augmentation. We applied similar camera augmentation to the lifting model, but did not see improvement in performance. We also show the results of pose retrieval using aligned 2D keypoints only. The poor performance of using input 2D keypoints for retrieval from different views confirms the fact that models must learn view invariance from inputs for this task.

Quantitative Results

From Table 5.1, we see that Pr-VIPE (with augmentation) outperforms all the baselines for H3.6M and 3DHP. The H3.6M results shown are on the hold-out set, and 3DHP is unseen during training, with more diverse poses and views. When we use all the cameras from 3DHP, we evaluate the generalization ability of models to new poses and new views. When we evaluate using only the 5 chest-level cameras from

Dataset	H3.6M			3DHP (Chest)			3DHP (All)		
k	1	10	20	1	10	20	1	10	20
2D keypoints*	28.7	47.1	50.9	5.20	14.0	17.2	9.80	21.6	25.5
3D lifting*	69.0	89.7	92.7	24.9	54.4	62.4	24.6	53.2	61.3
$L2$-VIPE	73.5	94.2	96.6	23.8	56.7	66.5	18.7	46.3	55.7
$L2$-VIPE (w/ aug.)	70.4	91.8	94.5	24.9	55.4	63.6	23.7	53.0	61.4
Pr-VIPE	**76.2**	**95.6**	**97.7**	25.4	59.3	69.3	19.9	49.1	58.8
Pr-VIPE (w/ aug.)	73.7	93.9	96.3	**28.3**	**62.3**	**71.4**	**26.4**	**58.6**	**67.9**

Table 5.1: Comparison of cross-view pose retrieval results Hit@k (%) on H3.6M and 3DHP with chest-level cameras and all cameras. $*$ indicates that normalization and Procrustes alignment are performed on query-index pairs.

3DHP, where the views are more similar to the training set in H3.6M, we mainly evaluate for generalization to new poses. When we evaluate using only the 5 chest-level cameras from 3DHP, the views are more similar to H3.6M, and generalization to new poses becomes more important. Our model is robust to the choice of β and the number of samples K (analysis in supplementary materials of [53]).

Table 5.1 shows that Pr-VIPE without camera augmentation is able to perform better than the baselines for H3.6M and 3DHP (chest-level cameras). This shows that Pr-VIPE is able to generalize as well as other baseline methods to new poses. However, for 3DHP (all cameras), the performance for Pr-VIPE without augmentation is worse compared with chest-level cameras. This observation indicates that when trained on chest-level cameras only, Pr-VIPE does not generalize as well to new views. The same results can be observed for $L2$-VIPE between chest-level and all cameras. In contrast, the 3D lifting models are able to generalize better to new views with the help of additional Procrustes alignment, which requires expensive SVD computation for every index-query pair.

We further apply camera augmentation to training the Pr-VIPE and the $L2$-VIPE model. Note that this step does not require camera parameters or additional groundtruth. The results in Table 5.1 on Pr-VIPE show that the augmentation improves performance for 3DHP (all cameras) by 6% to 9%. This step also increases chest-level camera accuracy slightly. For $L2$-VIPE, we can observe a similar increase on all views. Camera augmentation reduces accuracy on H3.6M for both models. This is likely because augmentation reduces overfitting to the training camera views. By performing camera augmentation, Pr-VIPE is able to generalize better to new poses and new views.

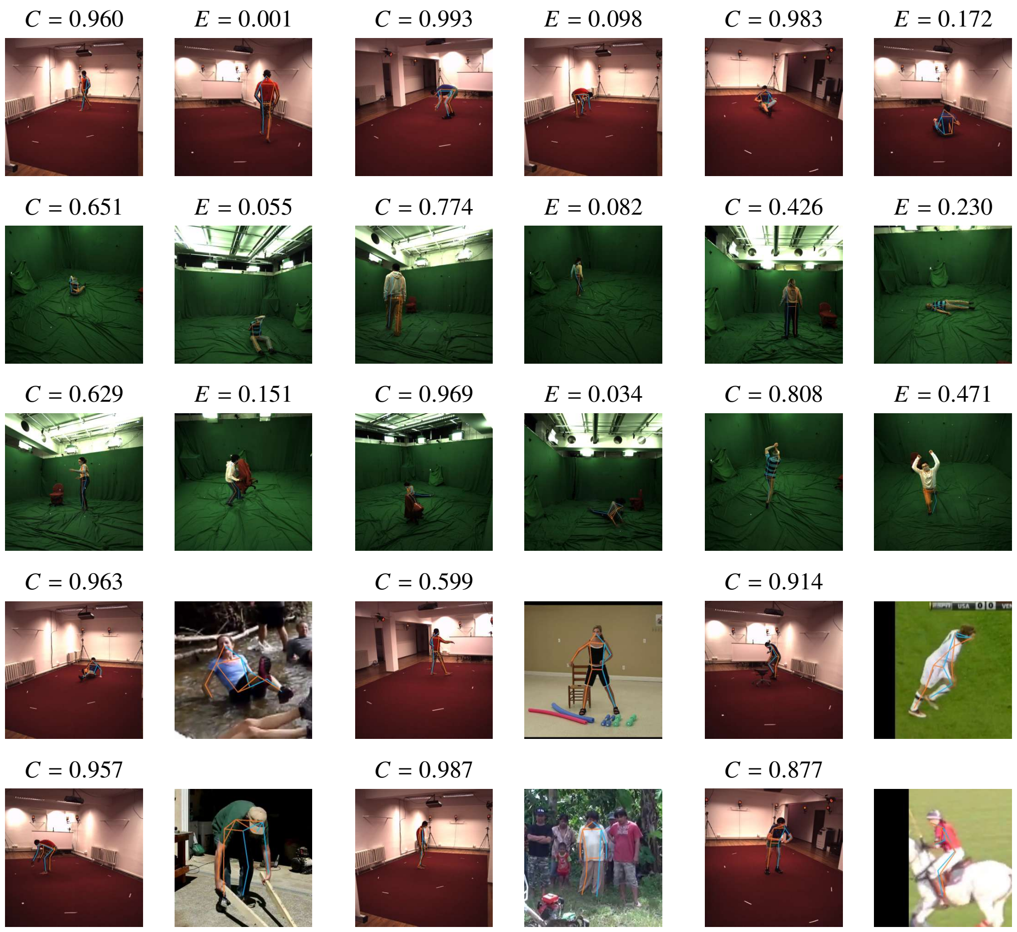


Figure 5.3: Visualization of pose retrieval results. The first row is from H3.6M; the second and the third row are from 3DHP; the last two rows are using queries from H3.6M to retrieve from 2DHP. On each row, we show the query pose on the left for each image pair and the top-1 retrieval using the Pr-VIPE model (w/ aug.) on the right. We display retrieval confidences ("C") and top-1 NP-MPJPEs ("E", if 3D pose groundtruth is available).

Qualitative Results

Fig. 5.3 shows qualitative retrieval results using Pr-VIPE. As shown in the first row, the retrieval confidence of the model is generally high for H3.6M. This indicates that the retrieved poses are close to their queries in the embedding space. Errors in 2D keypoint detection can lead to retrieval errors as shown by the rightmost pair. In the second and third rows, the retrieval confidence is lower for 3DHP. This is likely because there are new poses and views unseen during training, which has the nearest neighbor slightly further away in the embedding space. We see that the model can generalize to new views as the images are taken at different camera elevations from H3.6M. Interestingly, the rightmost pair on row 2 shows that the model can retrieve poses with large differences in roll angle, which is not present in the training set. The rightmost pair on row 3 shows an example of a large NP-MPJPE error due to mis-detection of the left leg in the index pose.

We show qualitative results using queries from the H3.6M hold-out set to retrieve from 2DHP in the last two rows of Fig. 5.3. The results on these in-the-wild images indicate that as long as the 2D keypoint detector works reliably, our model is able to retrieve poses across views and subjects. More qualitative results are provided in the supplementary materials of [53].

Downstream Tasks

We show that our pose embedding can be directly applied to pose-based downstream tasks using simple algorithms. We compare the performance of Pr-VIPE (**only trained on H3.6M, with no additional training**) on the Penn Action dataset against other approaches specifically trained for each task on the target dataset. In all the following experiments in this section, we compute our Pr-VIPE embeddings on single video frames and use the negative logarithm of the matching probability (5.7) as the distance between two frames. Then we apply temporal averaging within an atrous kernel of size 7 and rate 3 around the two center frames and use this averaged distance as the frame matching distance. Given the matching distance, we use standard dynamic time warping (DTW) algorithm to align two action sequences by minimizing the sum of frame matching distances. We further use the averaged frame matching distance from the alignment as the distance between two video sequences.

Action Recognition

We evaluate our embeddings for action recognition using nearest neighbor search with the sequence distance described above. Provided person bounding boxes in each frame, we estimate 2D pose keypoints using [42]. On Penn Action, we use the standard train/test split [37]. Using all the testing videos as queries, we conduct two experiments: (1) we use all training videos as index to evaluate overall performance and compare with state-of-the-art methods, and (2) we use training videos only under one view as index and evaluate the effectiveness of our embeddings in terms of view-invariance. For this second experiment, actions with zero or only one sample under the index view are ignored, and accuracy is averaged over different views.

From Table 5.2 we can see that without any training on the target domain or using image context information, our embeddings can achieve highly competitive results

Methods	Input			Accuracy (%)
	RGB	Flow	Pose	
Nie *et al.* [37]	✓		✓	85.5
Iqbal *et al.* [18]			✓	79.0
Cao *et al.* [5]		✓	✓	95.3
	✓	✓		98.1
Du *et al.* [10]	✓	✓	✓	97.4
Liu *et al.* [30]	✓		✓	91.4
Luvizon *et al.* [32]	✓		✓	98.7
Ours			✓	97.5
Ours (1-view index)			✓	92.1

Table 5.2: Comparison of action recognition results on Penn Action.

Methods	Kendall's Tau
SaL [35]	0.6336
TCN [52]	0.7353
TCC [11]	0.7328
TCC + SaL [11]	0.7286
TCC + TCN [11]	0.7672
Ours	0.7476
Ours (same-view only)	0.7521
Ours (different-view only)	0.7607

Table 5.3: Comparison of video alignment results on Penn Action.

on pose-based action classification, outperforming the existing best baseline that only uses pose input and even some other methods that rely on image context or optical flow. As shown in the last row in Table 5.2, our embeddings can be used to classify actions from different views using index samples from only one single view with relatively high accuracy, which further demonstrates the advantages of our view-invariant embeddings.

Video Alignment

Our embeddings can be used to align human action videos from different views using DTW algorithm as described earlier in Section 5.4. We measure the alignment quality of our embeddings quantitatively using Kendall's Tau [11], which reflects how well an embedding model can be applied to align unseen sequences if we use nearest neighbor in the embedding space to match frames for video pairs. A value of 1 corresponds to perfect alignment. We also test the view-invariant properties of our embeddings by evaluating Kendall's Tau on aligning videos pairs from the same view, and aligning pairs with different views.

In Table 5.3, we compare our results with other video embedding baselines that are trained for the alignment task on Penn Action, from which we observe that

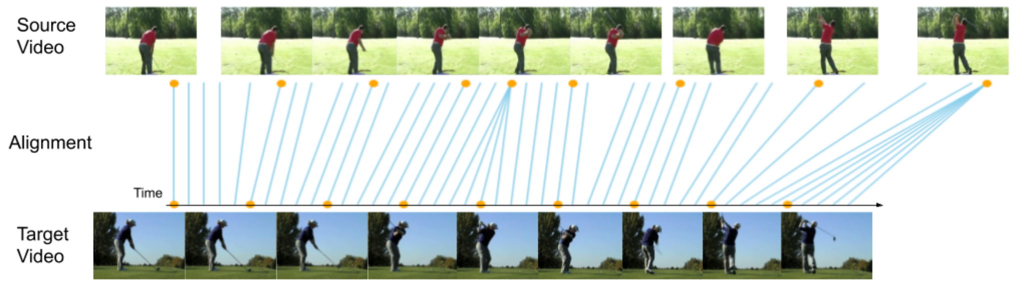


Figure 5.4: Video alignment results using Pr-VIPE. The orange dots correspond to the visualized frames, and the blue line segments illustrate the frame alignment.

Pr-VIPE performs better than all the method that use a single type of loss. While Pr-VIPE is slightly worse than the combined TCC+TCN loss, our embeddings are able to achieve this without being explicitly trained for this task or taking advantage of image context. In the last two rows of Table 5.3, we show the results from evaluating video pairs only from the same or different views. We can see that our embedding achieves consistently high performance regardless of whether the aligned video pair is from the same or different views, which demonstrate its view-invariant property. In Fig. 5.4, we show action video synchronization results from different views using Pr-VIPE. We provide more synchronized videos for all actions in the supplementary materials of [53].

5.5 Extensions to Temporal Embeddings

For understanding actions, sequences of human poses are usually required as they provide important temporal information. In this section, we further extend the Pr-VIPE framework to temporal domain, namely Temporal Pr-VIPE, to explicitly handle sequential inputs. Instead of embedding a single 2D pose, we embed a 2D pose sequence with the view-invariance and probabilistic properties of Pr-VIPE.

The input to our Temporal Pr-VIPE is a sequence of 2D poses from T temporally-ordered frames. Atrous sampling is used with a rate based on the video frame rate. We then apply the full Pr-VIPE objective to train an embedding model for 2D pose sequences. To compare whether a pair of pose sequences are similar, we compute the NP-MPJPEs of the 3D poses between each of their corresponding frame pairs and threshold on the maximum pairwise NP-MPJPE. We also apply camera augmentation, similar to the Pr-VIPE training, by applying a random camera view to a subset of sequences within each batch during training.

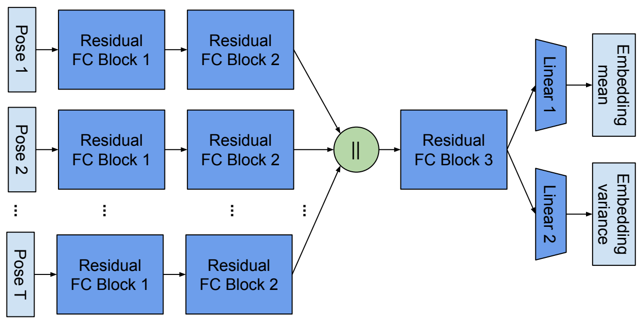


Figure 5.5: Temporal Pr-VIPE model architecture. The green circle represents vector concatenation.

Dataset	Dim.	H3.6M				3DHP (Chest)				3DHP (All)			
k		1	5	10	20	1	5	10	20	1	5	10	20
Pr-VIPE (stacking 8D)	56	70.3	85.4	89.8	93.1	41.9	60.6	67.8	74.7	38.9	56.9	64.3	71.3
Pr-VIPE (stacking 16D)	112	78.7	91.1	94.3	96.6	48.9	67.4	74.3	80.5	45.0	63.2	70.2	76.8
Pr-VIPE (stacking 32D)	224	80.8	92.8	95.8	97.7	50.4	69.3	76.2	82.4	47.0	65.4	72.4	78.9
Temporal Pr-VIPE (16D)	16	72.2	87.9	91.9	94.9	39.5	60.8	68.7	75.7	36.8	57.8	65.9	73.2
Temporal Pr-VIPE (32D)	32	80.0	92.1	95.0	97.1	49.2	69.1	75.7	81.3	46.7	66.4	73.3	79.3
Temporal Pr-VIPE (56D)	56	80.4	92.3	95.1	97.1	50.4	70.1	76.4	82.0	47.7	67.2	73.9	79.8

Table 5.4: Comparison of cross-view pose sequence retrieval results Hit@k (%) on H3.6M and 3DHP. All the models in the table use camera augmentation. The "Dim." column refers to the total dimensionality of the embeddings.

We adopt a mid-fusion style network model architecture. Each 2D pose from the input sequence is fed into a network with two residual FC blocks [33], and the output features are concatenated and fed to a third residual FC block followed by linear heads for final predictions, as shown in Fig. 5.5. More details on the implementation can be found in [31].

Temporal Experiments

We evaluate our Temporal Pr-VIPE models for cross-view pose sequence retrieval. Similar to cross-view pose retrieval, this task targets evaluating temporal embedding quality in terms of retrieving pose sequences to match a query sequence from a different camera view.

Evaluation Procedure. Given two pose sequences of the same length, we first compute the NP-MPJPE between each corresponding pose pair from both sequences. Then the maximum of these pairwise NP-MPJPEs is defined as the NP-MPJPE between the two sequences. We choose maximum here to reflect our requirement for the two sequences to be strictly close at all timestamps.

For each query sequence, we retrieve its k nearest neighbor sequences from an index set based on their embedding distances. Then the sequence NP-MPJPE between each query and retrieval pair is thresholded to determine whether each retrieval is a correct match. Similar to the single-frame pose retrieval task, we evaluate on the H3.6M and the 3DHP dataset. We iterate through all camera pairs in each dataset as query and index, and report averaged results across all such camera pairs.

We compare our Temporal Pr-VIPE result with baseline methods that stack single-frame embeddings within the same frame window into higher dimensional embeddings for retrieval distance computation. As we shall show in Section 5.5, simple stacking is an effective way of combining frame-level embeddings for sequence retrieval. However, it comes with a major drawback of high embedding dimensions, which can be prohibitive for large-scale applications. In this experiment, we demonstrate that with Temporal Pr-VIPE, we are able to achieve competitive retrieval performance with a much smaller embedding dimension.

Quantitative Results. From Table 5.4, we see that using Temporal Pr-VIPE, we are able to achieve competitive results with a much lower embedding dimension. Specifically, our 32D temporal Pr-VIPE outperforms stacking 7 8D single-frame embeddings (total 56D) by a large margin. It also achieves slightly better performance compared to stacking 7 16D embeddings (total 112D), with fewer than one third of the embedding dimensions. We observe a similar trend across both H3.6M and 3DHP (unseen test set with new poses and new views), suggesting that both models have similar generalization abilities to new poses and new views. When we vary the number of dimensions for temporal Pr-VIPE, we note that at least 32D embeddings is needed to achieve comparable performance to stacking 16D Pr-VIPE (total 112D), and at least 56D embeddings is needed to achieve comparable performance to stacking 32D Pr-VIPE (total 224D). Additionally, we note that when the output dimensions are comparable, at total 56D, Temporal Pr-VIPE performs much better than stacking 7 8D Pr-VIPE.

Pr-VIPE has a simple architecture and can be potentially applied to other domains, such as hand pose or other generic object pose recognition. With this work, we hope to encourage further explorations into approaching pose related problems from an embedding perspective, especially where recognizing 3D similarity is central to the problem.

References

[1] Ijaz Akhter and Michael J. Black. “Pose-Conditioned Joint Angle Limits for 3D Human Pose Reconstruction.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2015.

[2] Mykhaylo Andriluka et al. “2D Human Pose Estimation: New Benchmark and State of the Art Analysis.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2014.

[3] Aleksandar Bojchevski and Stephan Günnemann. “Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking.” In: *International Conference on Learning Representations*. 2018.

[4] Jane Bromley et al. “Signature Verification Using a “Siamese” Time Delay Neural Network.” In: *Advances in Neural Information Processing Systems*. 1994.

[5] Congqi Cao et al. “Body Joint Guided 3-D Deep Convolutional Descriptors for Action Recognition.” In: *IEEE Transactions on Cybernetics* 48.3 (2017), pp. 1095–1108.

[6] Ching-Hang Chen and Deva Ramanan. “3D Human Pose Estimation = 2D Pose Estimation + Matching.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7035–7043.

[7] Ching-Hang Chen et al. “Unsupervised 3D Pose Estimation with Geometric Self-Supervision.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5714–5724.

[8] Ruihang Chu et al. “Vehicle Re-Identification with Viewpoint-Aware Metric Learning.” In: *International Conference on Computer Vision*. 2019.

[9] Dylan Drover et al. “Can 3D Pose Be Learned from 2D Projections Alone?” In: *European Conference on Computer Vision*. 2018.

[10] Wenbin Du, Yali Wang, and Yu Qiao. “RPAN: An End-to-End Recurrent Pose-Attention Network for Action Recognition in Videos.” In: *International Conference on Computer Vision (ICCV)*. 2017.

[11] Debidatta Dwibedi et al. “Temporal Cycle-Consistency Learning.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.

[12] Raia Hadsell, Sumit Chopra, and Yann LeCun. “Dimensionality Reduction by Learning an Invariant Mapping.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2006.

[13] Alexander Hermans, Lucas Beyer, and Bastian Leibe. “In Defense of the Triplet Loss for Person Re-Identification.” In: *arXiv:1703.07737* (2017).

[14] Chih-Hui Ho et al. “PIEs: Pose Invariant Embeddings.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12377–12386.

[15] Wenze Hu and Song-Chun Zhu. “Learning a Probabilistic Model Mixing 3D and 2D Primitives for View Invariant Object Recognition.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2010.

[16] Chen Huang, Chen Change Loy, and Xiaoou Tang. “Local Similarity-Aware Deep Feature Embedding.” In: *Advances in Neural Information Processing Systems*. 2016.

[17] Catalin Ionescu et al. “Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.7 (2013), pp. 1325–1339.

[18] Umar Iqbal, Martin Garbade, and Juergen Gall. “Pose for Action–Action for Pose.” In: *The IEEE Conference on Automatic Face and Gesture Recognition (FG)*. 2017.

[19] Ahmet Iscen et al. “Mining on Manifolds: Metric Learning Without Labels.” In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[20] Karim Iskakov et al. “Learnable Triangulation of Human Pose.” In: *The IEEE International Conference on Computer Vision (ICCV)*. 2019.

[21] Nataraj Jammalamadaka et al. “Video Retrieval by Mimicking Poses.” In: *ACM International Conference on Multimedia Retrieval (ICMR)*. 2012.

[22] Xiaofei Ji and Honghai Liu. “Advances in View-Invariant Human Motion Analysis: A Review.” In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.1 (2009), pp. 13–24.

[23] Xiaofei Ji et al. “Visual-Based View-Invariant Human Motion Analysis: A Review.” In: *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer. 2008, pp. 741–748.

[24] Alex. Kendall and Yarin Gal. “What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?” In: *Advances in Neural Information Processing Systems*. 2017.

[25] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes.” In: *International Conference on Learning Representations*. 2014.

[26] Muhammed. Kocabas, Salih Karagoz, and Emre Akbas. “Self-Supervised Learning of 3D Human Pose Using Multi-View Geometry.” In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[27] Yann. LeCun, Fu Jie Huang, Leon Bottou, et al. “Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting.” In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2004.

[28] Junnan. Li et al. “Unsupervised Learning of View-Invariant Action Representations.” In: *Advances in Neural Information Processing Systems*. 2018.

[29] Jian Liu, Naveed Akhtar, and Mian Ajmal. “Viewpoint Invariant Action Recognition Using RGB-D Videos.” In: *IEEE Access* 6 (2018), pp. 70061–70071.

[30] Mengyuan Liu and Junsong Yuan. “Recognizing Human Actions as the Evolution of Pose Estimation Maps.” In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[31] Ting Liu*, Jennifer J. Sun*, Long Zhao, Jiaping Zhao, Liangzhe Yuan, Yuxiao Wang, Liang-Chieh Chen, Florian Schroff, and Hartwig Adam. “View-Invariant, Occlusion-Robust Probabilistic Embedding for Human Pose.” In: *International Journal of Computer Vision* 130.1 (2022), pp. 111–135. URL: `https://arxiv.org/pdf/2010.13321.pdf`.

[32] Diogo C. Luvizon, Hedi Tabia, and David Picard. “Multi-Task Deep Learning for Real-Time 3D Human Pose Estimation and Action Recognition.” In: *arXiv:1912.08077* (2019).

[33] Julieta Martinez et al. “A Simple Yet Effective Baseline for 3D Human Pose Estimation.” In: *International Conference on Computer Vision*. 2017.

[34] Dushyant Mehta et al. “Monocular 3D Human Pose Estimation in the Wild Using Improved CNN Supervision.” In: *International Conference on 3D Vision*. 2017.

[35] Ishan Misra, C. Lawrence Zitnick, and Martial Hebert. “Shuffle and Learn: Unsupervised Learning Using Temporal Order Verification.” In: *European Conference on Computer Vision*. 2016.

[36] Greg Mori et al. “Pose Embeddings: A Deep Architecture for Learning to Match Human Poses.” In: *arXiv:1507.00302* (2015).

[37] Bruce Xiaohan Nie, Caiming Xiong, and Song-Chun Zhu. “Joint Action Recognition and Pose Estimation from Video.” In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[38] Seong Joon Oh et al. “Modeling Uncertainty with Hedged Instance Embedding.” In: *International Conference on Learning Representations* (2019).

[39] Song Hyun Oh et al. “Deep Metric Learning via Lifted Structured Feature Embedding.” In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[40] Eng-Jon Ong et al. “Viewpoint Invariant Exemplar-Based 3D Human Tracking.” In: *Computer Vision and Image Understanding* 104.2-3 (2006), pp. 178–189.

[41] George Papandreou et al. “PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model.” In: *European Conference on Computer Vision*. 2018.

[42] George Papandreou et al. “Towards Accurate Multi-Person Pose Estimation in the Wild.” In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[43] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. “Deep Face Recognition.” In: *BMVC*. 2015.

[44] Dario Pavllo et al. “3D Human Pose Estimation in Video with Temporal Convolutions and Semi-Supervised Training.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7753–7762.

[45] Haibo Qiu et al. “Cross View Fusion for 3D Human Pose Estimation.” In: *International Conference on Computer Vision*. 2019.

[46] Cen Rao and Mubarak Shah. “View-Invariance in Action Recognition.” In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 2. IEEE. 2001, pp. II–II.

[47] Mir Imtiaz Hossain Rayat and James J. Little. “Exploiting Temporal Information for 3D Human Pose Estimation.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 68–84.

[48] Helge Rhodin, Mathieu Salzmann, and Pascal Fua. “Unsupervised Geometry-Aware Representation for 3D Human Pose Estimation.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 750–767.

[49] Helge Rhodin et al. “Learning Monocular 3D Human Pose Estimation from Multi-View Images.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8437–8446.

[50] Helge Rhodin et al. “Neural Scene Decomposition for Multi-Person Motion Capture.” In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[51] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A Unified Embedding for Face Recognition and Clustering.” In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[52] Pierre Sermanet et al. “Time-Contrastive Networks: Self-Supervised Learning from Video.” In: *IEEE International Conference on Robotics and Automation*. 2018.

[53] Jennifer J. Sun, Jiaping Zhao, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, and Ting Liu. "View-Invariant Probabilistic Embedding for Human Pose." In: *European Conference on Computer Vision (ECCV)* (2020), pp. 53–70. URL: `https://arxiv.org/pdf/1912.01001.pdf`.

[54] Xiao Sun et al. "Integral Human Pose Regression." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 529–545.

[55] Bugra Tekin et al. "Learning to Fuse 2D and 3D Image Cues for Monocular Body Pose Estimation." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3941–3950.

[56] Denis Tome et al. "Rethinking Pose in 3D: Multi-Stage Refinement and Recovery for Markerless Motion Capture." In: *2018 International Conference on 3D Vision (3DV)*. IEEE. 2018, pp. 474–483.

[57] Luke Vilnis and Andrew McCallum. "Word Representations via Gaussian Embedding." In: *International Conference on Learning Representations*. 2015.

[58] Jiang Wang et al. "Learning Fine-Grained Image Similarity with Deep Ranking." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2014.

[59] Paul Wohlhart and Vincent Lepetit. "Learning Descriptors for Object Recognition and 3D Pose Estimation." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2015.

[60] Chao-Yuan Wu et al. "Sampling Matters in Deep Embedding Learning." In: *International Conference on Computer Vision (ICCV)*. 2017.

[61] Lu Xia, Chia-Chih Chen, and Jake K. Aggarwal. "View Invariant Human Action Recognition using Histograms of 3D Joints." In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 2012.

[62] Weiyu Zhang, Menglong Zhu, and Konstantinos G. Derpanis. "From Actemes to Action: A Strongly-Supervised Representation for Detailed Action Understanding." In: *International Conference on Computer Vision (ICCV)*. 2013.

[63] Liang Zheng et al. "Pose-Invariant Embedding for Deep Person Re-Identification." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[64] Xingyi Zhou et al. "Towards 3D Human Pose Estimation in the Wild: A Weakly-Supervised Approach." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 398–407.

Chapter 6

BENCHMARKING REPRESENTATION LEARNING

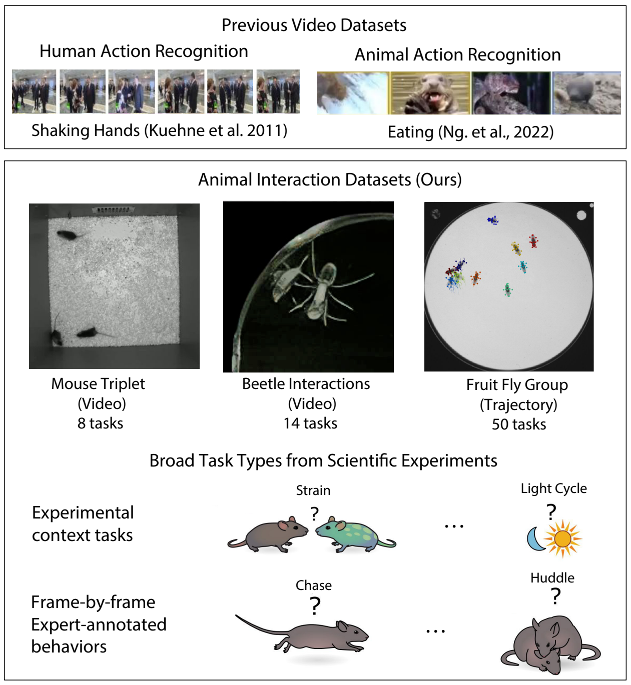

Previous Video Datasets
Human Action Recognition
Animal Action Recognition
Shaking Hands (Kuehne et al. 2011)
Eating (Ng. et al., 2022)
Animal Interaction Datasets (Ours)
Mouse Triplet (Video) 8 tasks
Beetle Interactions (Video) 14 tasks
Fruit Fly Group (Trajectory) 50 tasks
Broad Task Types from Scientific Experiments
Experimental context tasks
Strain
Light Cycle
Frame-by-frame Expert-annotated behaviors
Chase
Huddle

Figure 6.1: MABe22 consists of animal interactions in laboratory experiments. We propose a dataset to benchmark representation learning methods that focus on multi-agent behavior. Our benchmark includes a large video and trajectory library depicting interactions of mice, beetles, ants, and fruit flies alongside a large suite of downstream tasks to measure representation quality. Tasks differ across model organisms and include the classification of experimental conditions (e.g., species strain, light cycle, optogenetic activations, interaction duration) as well as expert-annotated actions (e.g., chase, huddle, and sniffs for mice).

This chapter is mainly based on the following paper:

[1] Jennifer J. Sun, Markus Marks, Andrew Ulmer, Dipam Chakraborty, Brian Geuther, Edward Hayes, Heng Jia, Vivek Kumar, Zachary Partridge, Alice Robie, Catherine E. Schretter, et al. "MABe22: A Multi-Species Multi-Task Benchmark for Learned Representations of Behavior." In: *International Conference on Machine Learning* (2023). URL: `https://arxiv.org/pdf/2207.10553.pdf`.

Abstract. We introduce *MABe22*, a large-scale, multi-agent video and trajectory benchmark to assess the quality of learned behavior representations. This dataset is collected from a variety of biology experiments, and includes triplets of interacting mice (4.7 million frames video+pose tracking data, 10 million frames pose only), symbiotic beetle-ant interactions (10 million frames video data), and groups of interacting flies (4.4 million frames of pose tracking data). Accompanying these data, we introduce a panel of real-life downstream analysis tasks to assess the quality of learned representations by evaluating how well they preserve information about the experimental conditions (e.g., strain, time of day, optogenetic stimulation) and animal behavior. We test multiple state-of-the-art self-supervised video and trajectory representation learning methods to demonstrate the use of our benchmark, revealing that methods developed using human action datasets do not fully translate to animal datasets. We hope that our benchmark and dataset encourage a broader exploration of behavior representation learning methods across species and settings.

6.1 Introduction

The study of interacting agents is important for a range of scientific and engineering applications, from designing safer autonomous vehicles [10], to understanding player behavior in virtual worlds [27], to uncovering the biological underpinnings of neurological disorders [54, 68]. Across disciplines, there is a need for new techniques to characterize the structure of multi-agent behavior with greater precision, sensitivity, and detail. Traditionally, behavior analysis models are trained with full supervision [7, 28, 5], which subjects users to a heavy burden of video annotation. Efforts to learn behavioral representations without manual annotation [3, 67, 29, 58] promise to bypass this labor bottleneck, but are difficult to evaluate systematically. To support the development of learned behavioral representations, and to better evaluate their performance, we need benchmark datasets for behavior. These benchmarks should cover a broad range of experimental conditions, to avoid overfitting on the statistics of a particular dataset. Furthermore, when representations are learned without supervision, there is no obvious metric to evaluate the quality of the representation. Yet, a metric is needed for quantitative comparisons. These two challenges inspired our work.

We have collected and curated a large dataset and benchmark from biology experiments for evaluating learned representations of social behavior (Figure 6.1). We chose to focus on videos of laboratory animals for several reasons:

- Animal behavioral experiments are collected against a uniform uninformative background, such as [54, 15, 49], and thus behavior classifiers are forced to focus on the dynamic and pictorial cues of the action. In contrast, video of human behavior, e.g., actions in different sports, are usually *pictorially informative*, meaning that the action itself can be classified from the appearance of a single or a few frames rather than considering motion over long periods of time.
- Animal behavior is often recorded under various experimental manipulations that impact the behavior (Figure 6.1). Identifying those experimental manipulations provides an objective task that may be used to evaluate the quality of a representation. This complements evaluation based on reproducing human annotations of behavior, which have shorter temporal structure but can be subjective [2].
- The biologists who provided us with videos of their experiments are engaged in analyzing specific aspects of the animals' behavior. Using a given representation to automate their analysis provides us with an objective performance criterion that is defined outside the field of Computer Vision. Evaluation methods based on *downstream tasks*, i.e., tasks where the representation is used to analyze specific aspects of the signal, have been used in other domains, e.g., for evaluating visual representations [63] or neural mechanistic models [53].
- Our dataset is from real-world neuroscience and evolutionary biology experiments, and progress on this dataset will enable biologists to use the representations generated to study how behavior changes as a function of other experimental variables.

We make three contributions: **1.** A large and richly annotated video and trajectory dataset, **M**ulti-**A**gent **Be**havior 20**22** (MABe22), of social behavior in three species: laboratory mice (*Mus musculus*) triplets, rove beetles (*Sceptobius lativentris*) paired with their symbiotic host species or with other beetles, and vinegar flies (*Drosophila melanogaster*). **2.** A large and diverse set of downstream evaluation tasks based on the classification of experimental conditions (optogenetic activation, animal strain, time-of-day) and expert-annotated behavior labels. **3.** A baseline benchmark of state-of-the-art self-supervised video and trajectory representation learning, as well as community-contributed methods solicited from an open challenge. To the best of our knowledge, our dataset is the first to provide non-annotation-based downstream tasks from scientific experiments for representation evaluation (Table 6.1).

Dataset	Number of species	Annotation frequency	Action classes	Downstream tasks	Size
Kinetics400 [33]	1 (human)	clip	400	x	306k clips
HMDB [39]	1 (human)	clip	51	x	6776 clips
UCF [56]	1 (human)	clip	101	x	13320 clips
Animal Kingdom [47]	850	frame	140	x	4.5M frames
CalMS21 [59]	1	frame	7	x	1M frames +6M unlabelled
Fly vs. Fly [15]	1	frame	10	x	1.5M frames
CRIM13 [7]	1	frame	13	x	8M frames
Our Dataset	4	frame	16	**56** from experiments	15M frames video + 14M frames traj

Table 6.1: Comparison with commonly used, public video and trajectory datasets. While existing datasets can be used for behavioral representation learning, the downstream evaluation focuses on a single type of task (detection and classification of human-annotated actions) or a single species. Our benchmark introduces a rich set of downstream analysis tasks that we obtain from scientific experiments on multiple species.

6.2 Related Work

Related Animal Datasets. The goal of the MABe22 dataset is to benchmark representation learning models for behavior analysis using data from biology experiments. There are several existing datasets for studying animal social behavior, including CRIM13 [7], Fly vs. Fly [15], and CalMS21 [59]. These datasets contain video or pose data from interacting animals, as well as human-annotated behavior labels (Table 6.1); they all focus on a single species and setting. AnimalKingdom [47] is another recent animal behavior dataset that includes social and nonsocial behavior from multiple species, but is focused on human annotation-based action recognition only. Our dataset is unique in that it defines a range of downstream tasks for each organism; these tasks are motivated by scientific experiments, with the goal of to driving scientific discovery in biology.

Related Human Datasets. While animal video datasets remain comparatively rate, there are many video datasets designed for work in human action recognition. Human datasets typically have very different visual characteristics from animal datasets. Most notably, many human datasets that are used to benchmark self-supervised video representation learning, such as Kinetics [33], UCF101 [56] and HMDB51 [39], contain “spatially heavy” visual information that informs downstream action classification — that is, different actions have different backgrounds. Because of these differences in the visual appearance, agents’ actions can be partly distinguished by these visual features alone, without models having to learn any temporal features of the agents' behavior. In contrast, our animal videos are all acquired against a stationary, neutral background, forcing models to use the temporal structure of the data to distinguish between actions.

Related Problems in Multi-Agent Behavior. While our dataset is composed of multi-agent data from biology, there are also multi-agent behavior datasets from other domains, such as from autonomous driving [10, 61], sports analytics [69, 13], and video games [51, 22]. These datasets often focus on forecasting, motion planning, and reinforcement learning, whereas our dataset is used for tasks from scientific applications, such as distinguishing animal strains via observed behaviors.

Work in Animal Behavior Analysis. In biology and neuroscience, computational models of behavior have the potential to significantly reduce human data annotation efforts, and to provide more detailed descriptions of the behavior in question [2, 49]. Automated characterizations of animal behavior have been used to study the relationship between neural activity and behavior [42], to characterize behavioral differences between species and between different strains within a species [26], and to quantify the effect of functional or pharmacological perturbations [50, 68]. The input to these models may be video [5] or trajectory data [58, 54].

Supervised behavior models have been trained to identify human-defined behaviors-of-interest [28, 54, 43, 32], often using frame-by-frame behavior annotations from domain experts. Another body of work discovers behaviors without human annotations, using unsupervised and self-supervised methods [3, 67, 29, 41, 8] that learn the latent structure of behavioral data. The learned representation may be continuous [58], or discrete, such as when discovering behavior motifs [3, 67, 29]. There currently does not exist a unified behavioral representation learning dataset that can compare these models across a broad range of behavior analysis settings. Here, we propose MABe 2022 for evaluating the performance of these representation learning methods.

Work in Representation Learning. Representation learning for visual [19, 11, 48, 38, 23] and trajectory data [58, 70] has been applied to a variety of tasks, such as for image classification [11], speech recognition [48], and behavior classification [58]. In these works, many different unsupervised / self-supervised methods have been developed, employing various pretext tasks to pre-train a model, such as classifying image rotations [19], predicting future observations [48], contrastive learning with image augmentations [11], and decoding programmatic attributes [58]. The quality of learned representations is often evaluated on downstream tasks.

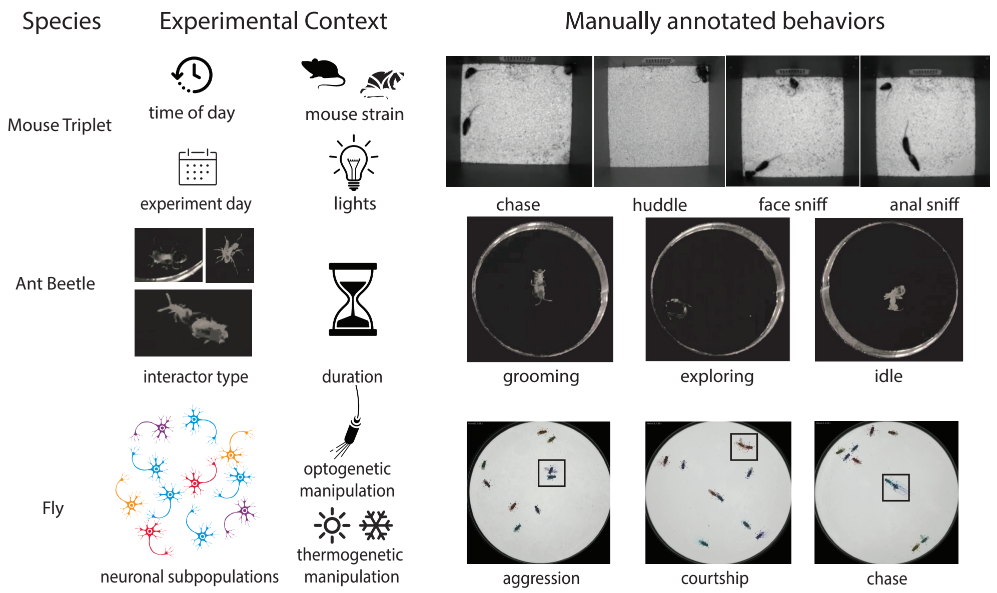


Figure 6.2: Summary of tasks and actions in our dataset. Our dataset includes three different species: mice, beetles with an intractor (an ant or other another beetle), and flies. The mouse dataset has both video and trajectory available, the beetle dataset is video-based, and the fly dataset is trajectory based. Classification of experimental conditions is used as a performance metric (examples depicted on the left for each dataset). Additionally, we collected conventionally expert-annotated actions (examples depicted on the right for each dataset), with frame-by-frame labels, e.g., as “chase”, “huddle”, “face sniff”, and “anogenital sniff” for mice. Overall, there are 72 behavior analysis tasks: 8 for mice, 14 for beetles and 50 for flies.

Behavioral Representation Learning. For behavior analysis, applications of representation learning include discovering behavior motifs [3, 67, 29, 41], identifying internal states [8], and improving sample-efficiency of supervised classifiers [58]. These works use methods such as variational autoencoders [34], autoregressive hidden Markov models [67], and Uniform Manifold Approximation and Projection (UMAP) [45] to characterize the latent structure of behavior. Notably, many groups have proposed methods for unsupervised behavior discovery [3, 36, 67, 41, 29, 44]. These works use different methods to model the temporal structure of behavior, including wavelet transforms [3], autoregressive hidden Markov models [67], and recurrent NNs [41], as well as different methods for segmenting behavior, such as Gaussian mixture models [29], k-means clustering [41], and watershed transforms [3]. Our goal is to develop a standardized dataset for evaluating these methods on a common set of behavior analysis tasks.

6.3 Dataset Design and Collection

We designed and curated MABe22, a multi-agent behavior dataset for the purpose of studying behavioral representation learning. Our dataset consists of data from multiple model organisms in neuroscience/biology: mice, beetles, and flies. For each dataset, we constructed a collection of tasks based on real-world scientific applications, including determining the experimental context of the organisms and capturing expert-annotated behaviors. There are 72 tasks in total: 8 for mice, 14 for beetles, and 50 for flies. For the purpose of establishing a benchmark, we define a "good" learned representation of animal behavior that can decode biologically meaningful hidden labels as well as annotations by experts. Some tasks apply to all frames of the recording (strain of mice), but not all tasks are apply to all frames (sniffing, since experts may annotate only a subset of the videos). More details are available in the datasheet for our dataset in [57].

The mouse dataset (Section 6.3) consists of 2614 clips of video and trajectory data (1 minute each at 30 Hz) curated from longer videos of a triplet of interacting mice over multiple recording days. The video and trajectory datasets are from the same clips, and the mice are tracked using [55]. We additionally release a larger set of 5336 clips of trajectory data for evaluating community-contributed methods. The beetle dataset (Section 6.3) consists of 11536 clips of video (30 seconds each at 30 Hz) curated from paired interactions of rove beetles (*Sceptobius lativentris*) with intact or manipulated members of their symbiotic host species, the velvety tree ant (*Liometopum occidentale*), or with other beetle species. The fly dataset (Section 6.3) consists of 968 clips of trajectory data (30-second clips at 150 Hz) of groups of 8-11 interacting flies, tracked using [31].

Mouse Triplets

Data Description. The mouse dataset consists of a set of videos and trajectories from three interacting mice, recorded from an overhead camera in an open field arena measuring 52cm x 52cm, with a grate located at the northern wall of the arena giving access to food and water. Animals were introduced to the arena one by one over the first ten minutes of recording and were recorded continuously for four days at a framerate of 30 Hz and a camera resolution of 800 x 800 pixels. Illumination was provided by an overhead light on a 24-hour reverse light cycle (lights off during the day and on at night); mice are nocturnal and thus are most active during the dark. Behavior was recorded using an IR-pass filter so that light status could not be detected by the eye in the recorded videos. Animals' posture was tracked using a pose estimation model [55] based on HRNet [60] with an identity embedding network to track long-term identity.

Tasks. Representations of the mouse dataset are evaluated on 8 tasks that capture information about animals' genetic background, environment, and expert-annotated behaviors. These tasks were selected based on their relevance to common scientific applications such as identifying the behavioral effects of differences in animals' genetic backgrounds or experimenter-imposed changes in their environment. We examined capacity of learned representations to determine animal strain, as well as environmental factors such as whether room lights were on or off (a proxy for day/night cycles, which modulate animal behavior). We also included two tasks to predict the day of the trajectory relative to the start of recording (animal behavior changes across days as they habituate to a new environment [37]), and the time of day of the trajectory (animal behavior changes over the course of a day, driven by circadian rhythms). A learned representation of behavior should also be rich enough to recapitulate human-produced labels of animals' moment-to-moment actions. Therefore our evaluation tasks include the detection of expert-annotated behaviors: huddling, chasing, face sniffing, and anogenital sniffing. A detailed description of the tasks is listed in [57].

Beetle Interactions

Dataset description. The beetle dataset consists of a rove beetle (*Sceptobius lativentris*) interacting one-on-one with its host ant (*Liometopum occidentale*), manipulated host ant (e.g., with pheromones stripped off) or with other insects (e.g., a nitidulid beetle). The original experiment consisted of two-hour interaction trials, from which we extracted a collection of 30-second clips. These recordings were made in 8-well behavioral interaction chambers (2cm diameter circles) in the dark and illuminated with inferred lights from the side/top. A top-mounted machine vision camera sensitive to IR light monitored the two-hour behavioral trials at 60 Hz. For this dataset, individual circular wells were cropped/parsed from the multi-well video and saved at 800x800 resolution with downsampling to 30 Hz.

Tasks. The beetle dataset includes tasks based on environmental conditions as well as expert-annotated behaviors. Labels for environmental conditions include the interactor type (the species of insect the rove beetle interacts with, and any experimental manipulations applied) as well as how long into the two-hour assay the observed clip occurred. The interactors represent a range of cue types, from the host organism with which the beetle should interact extensively to other insects that the beetle will likely ignore. We also provide expert annotations for six behaviors across the seven different types of one-on-one interactions. Generating a meaningful representation that extracts information of interest about the different behaviors adopted by the beetle in response to these disparate cues is crucial for insight into how species interact in nature. Details about the interaction tasks are described in [57].

Fly Groups

Data Description. The fly dataset consists of trajectories of groups of 8 to 11 vinegar flies (*Drosophila melanogaster*) interacting in a 5cm-diameter dish. The trajectories were derived from 96 videos of length 50k-75k frames, collected at 1024x1024 pixels and 150 frames per second. The flies' bodies and wings were tracked using FlyTracker [15], and landmarks on the body were tracked using the Animal Part Tracker (APT) [31] producing a total of 19 keypoints per tracked animal.

As the brain controls behavior, a good representation of behavior should change with neural activity. Thanks to its tractable genetics, precise neural activity manipulations are straightforward in *Drosophila*. We thus chose to perform experiments using optogenetic (light-activated neural activity via Chrimson) [35] and thermogenetic (heat activated, via TrpA) [50] activation of selected sets of neurons. We chose neurons (and the associated GAL4 lines) previously identified as controlling social behaviors, including courtship, avoidance [50], and female aggression [52]. For thermogenetic experiments, neural activation is constant and continuous for the entire video. Our optogenetic experiments consisted of activation for short periods of time at weak and strong intensities interspersed with periods of no activation. We combined these neural manipulations with genetic mutations and rearing conditions. Specifically, we selected populations of flies with the norpA mutation, which induces blindness [4], and either raised groups of flies together or separated by sex.

Tasks. The representations of the fly dataset are evaluated on a set of 50 tasks. Many of these tasks differentiate which populations of neurons are activated and how they are activated. For example, Task 5 indicates the activation of courtship neurons targeted by the R71G01 GAL4 line in groups of 5 male and 5 female flies. Task 31 compares how neurons were activated – it compares strong and weak activation of aIPg neurons, which regulate female aggression. Besides neural activation, tasks also differentiate flies based on sex, how the flies were raised, which strain they are from, and genetic mutations. A full list of tasks and the types of flies used are in [57].

Besides biological differences, we also include tasks based on manual annotations of the flies' behavior for the following social behaviors: any aggressive behavior toward another fly, chasing another fly, any courtship behavior toward another fly, high fencing, wing extension, and wing flick. We annotated behaviors sparsely across all videos with human experts using JAABA [32], with the goal of including annotations in a wide variety of flies and videos.

6.4 Benchmarking and Methods

We study how well behavioral representations generated by state-of-the-art self-supervised video representation learning methods are suited for decoding our hidden downstream biological tasks and human annotations (Section 6.4). We also solicit community-contributed methods for video and trajectory representation learning through an open competition (Section 6.4). The representation learned by the models is a mapping from each video frame/trajectory entry to a lower dimensional vector of fixed size. Here, we assume the evaluation tasks are hidden during representation learning. We then use this representation of the data to train a linear model to classify or regress to target values of the hidden downstream task, see supplementary materials of [57] for more details.

Self-supervised Video Representation Learning

Self-supervised video representation learning methods rely on designing pretext tasks that make use of prior knowledge about spatial and temporal information in videos to design pretext tasks such as temporal coherence [20], temporal ordering [46], the motion of an object [1], future prediction [65]. Contrastive learning [11, 25] has been used for learning good visual representations for instance discrimination. Another line of work has been introducing methods that solely rely on positive samples [21, 9]. In a recent comparison, the video version of Bootstrap Your Own Latent (BYOL) [21] has been shown to perform very well on the classic human benchmarks [16], with increased performance for an increased number of positive samples.

Masked Visual Modeling. Transformers [64] set the state-of-the-art across many AI fields, bridging language and vision models. Inspired by pretext tasks for language transformer models, such as masking in BERT [14], [24] recently introduced the Masked Auto-Encoder (MAE) for images, an effective pre-training method, by which an image is split into patches, and about 70 percent of the patches are masked. Based on the remaining patches, the task for the transformer is to reconstruct the masked patches. [17, 62] extended this framework to video, demonstrating transformers can be effectively pre-trained by masking 90 percent of the spatio-temporal volume. MaskFeat [66] showed that using HOG features [12] as reconstruction targets of masked patches is an effective pre-text task.

Community-Contributed Methods

In addition to studying state-of-the-art methods, our benchmarking efforts include community-contributed methods from an open competition. Our competition was hosted in two stages, where stage 1 consisted of the trajectory datasets from mouse and fly, and stage 2 consisted of video datasets from mouse and beetle. The test sets were private during the competition phase, and are now released as part of MABe22. We obtained around 1500 submissions in total at the end of the competition, and we summarize the top-performing method for the mouse, fly, and beetle datasets from this process for both video and trajectory data, with details for all methods in supplementary material of [57].

6.5 Experiments

We perform a large set of experiments to evaluate the performance of representation learning methods on MABe 2022 (Sections 6.5, 6.5). As video representation methods are more common, we focus on state-of-the-art video representation learning methods in this section. We additionally compare both community contributed video and trajectory representation learning methods. For each video representation learning method, we perform an ablation study on the key hyperparameter for the respective method and its effect on downstream task performance (Sections 6.5, 6.5), as well as pre-training on human datasets (Section 6.5). Finally, we present results from community-contributed methods on all datasets (Section 6.5), with additional results for the trajectory methods in supplementary material of [57].

Evaluation Procedure

From an input sequence of video/trajectory data of N frames ($N = 1800$ for mice and 4500 for flies), we evaluate models that produce learned representations of size $N \times D$, where D is the dimensionality of the representations. For video representation learning models, we use $D = 128$. For trajectory methods, we use $D = 128$ for mice and $D = 256$ for flies. We then use these feature vectors or

Mouse Triplets	Exp. Day ↓	Time of Day ↓	Strain ↑	Lights ↑	Manual Behaviors↑
ρBYOL (R-50 (Slow) 8x8) [16]	.0152	.0913	.9997	.9701	0.1832
Maskfeat (MViTv2-S 16x4) [66]	.0393	.0948	.9925	.7309	0.1627
MAE (ViT-B 16x4) [17]	**.0102**	**.0816**	**1.0000**	**.9758**	0.2309
(pretrained) ρBYOL (R-50 (Slow) 8x8)	.0176	.0910	.9994	.7967	**0.2688**
(pretrained) Maskfeat (MViTv2-S 16x4)	.0456	.0889	.9998	.7892	0.1896
(pretrained) MAE (ViT-B 16x4)	.0218	.0925	**1.0000**	.9391	0.2301

Ant Beetle	Duration ↓	Interactor Type ↑	Manual Behaviors ↑	Manual Behaviors (same) ↑
ρBYOL (R-50 (Slow) 8x8) [16]	**.0257**	.9999	.6178	.6457
Maskfeat (MViTv2-S 16x4) [66]	.0291	**1.0000**	.6212	.6574
MAE (ViT-B 16x4) [17]	.0283	**1.0000**	.6444	.6874
(pretrained) ρBYOL (R-50 (Slow) 8x8)	.0300	.9981	**.6967**	**.7334**
(pretrained) Maskfeat (MViTv2-S 16x4)	.0297	.9999	.6057	.6463
(pretrained) MAE (ViT-B 16x4)	.0300	.9999	.6879	.7077

Table 6.2: Evaluating self-supervised video representation learning methods. We evaluate representation learning performance using the linear evaluation protocol on downstream biologically relevant tasks. (pretrained) indicates pre-training on Kinetics400. ↓ indicates MSE and ↑ indicates F1 score. Mouse manual behaviors consist of chase, huddle, face sniff, anal sniff. Beetle manual behaviors consist of grooming, exploring, and idle, either for self (beetle) only or with the interactor. The best-performing model is in bold.

embeddings as inputs for a linear model that is used to classify/regress the hidden task. We use linear least squares with l2 regularized (Ridge) classification/regression as model and F1/mean-squared-error (MSE) as evaluation metrics (See more details in supplementary of [57]).

We evaluate a set of state-of-the-art video representation learning methods on MABe 2022, including Masked Autoencoder (MAE) [17] with a ViT-B backbone [64], MaskFeat [66] with a MViTv2-S backbone [40] and ρBYOL [16] with a SlowFast backbone (Slow pathway 8x8) [18]. We trained each method on our mice and beetle data, respectively, as well as used backbones pre-trained on human kinetics 400 [33].

Video Representation Results

We compare the performance of video representation learning methods on the mouse and beetle video datasets (Table 6.2). We find that the pre-trained ρBYOL (R-50 (Slow Pathway) 8x8 model performs best for all action recognition tasks (Manuel Behaviors). For all other downstream tasks training, a ViT-B 16x4 Masked Autoencoder (MAE) that is not pre-trained on Kinetics400 generally performs the best. This top performing MAE architecture uses spatio-temporal agnostic masking,

Mice Triplet	Exp. Day ↓	Time of Day ↓	Strain ↑	Lights ↑	Manual Behaviors↑
MAE Frame	.0239	.0886	1.000	.9525	.2020
MAE Cube	.0102	**.0816**	1.000	.9758	**.2309**
MAE Tube	**.0072**	.0835	1.000	**.9846**	.2249
Ant Beetle	**Duration ↓**	**Interactor Type ↑**	**Manual Behaviors ↑**	**Manual Behaviors (same) ↑**	
MAE Frame	.0301	.9999	.6169	.6497	
MAE Cube	**.0283**	**1.0000**	**.6444**	**.6874**	
MAE Tube	.0285	**1.0000**	.5802	.6351	

Table 6.3: Effect of masking strategy on MAE [17] performance. We evaluate different masking strategies (spatiotemporal random/cube, temporal/tube and spatial/frame) on the video datasets of MABe2022. For the mouse dataset cube/tube masking perform best, whereas for the beetle dataset cube/frame masking perform best. ↓ indicates MSE and ↑ indicates F1 score. The best-performing model is in bold.

Mice Triplet	Exp. Day ↓	Time of Day ↓	Strain ↑	Lights ↑	Manual Behaviors↑
2BYOL	.0298	**.0882**	.9994	.9588	**.1929**
3BYOL	.0225	.0906	.9983	.9492	.1733
4BYOL	**.0152**	.0913	.9997	**.9701**	.1771
Ant Beetle	**Duration ↓**	**Interactor Type ↑**	**Manual Behaviors ↑**	**Manual Behaviors (same) ↑**	
2BYOL	**.0237**	**1.0000**	.5943	.6498	
3BYOL	.0246	**1.0000**	**.6249**	**.6549**	
4BYOL	.0257	.9999	.6178	.6457	

Table 6.4: Effect of ρ on BYOL [16] performance. We evaluated the effect of the number of randomly sampled positives for ρBYOL. We find that for beetle 3 positive samples consistently have the best performance, while for mice, either 2 or 4 positives perform best depending on the task. ↓ indicates MSE and ↑ indicates F1 score. The best-performing model is in bold.

which likely performs well due to the observation that our datasets have very different spatio-temporal dynamics from each other and even more so from human datasets. We further discuss this in Section 6.5. We notice that the model that performs best for human annotated behaviors does not necessarily perform best for our downstream tasks that are based on experimental conditions. This indicates that models that pick up features that are most relevant for human perception and behavior definitions may not necessarily be the most informative features for other tasks.

Mice Triplet	Exp. Day ↓	Time of Day ↓	Strain ↑	Lights ↑	Manual Behaviors ↑	
BEiT + Hand-crafting	**.0093**	.0926	**1.0000**	**.9471**	.2603	
Vision Ensemble	.0441	.0922	.9832	.8048	**.2750**	
Multimodal MoCo/SimCLR	.0394	**.0912**	.9902	.7780	.2355	
Trajectory-BERT	.0932	.0996	.7202	.6729	.2379	
Ant Beetle	**Duration ↓**	**Interactor Type ↑**	**Manual Behaviors ↑**	**Manual Behaviors (same) ↑**		
BEiT + Hand-crafting	.0277	.9977	.6761	.7179		
Vision Ensemble	.0295	.9636	.6277	.6695		
Multimodal MoCo/SimCLR	**.0262**	**.9998**	**.7299**	**.7577**		
Fly Group	**Fly Type ↑**	**Stimulation, Control ↑**	**Stimulation, Aggression ↑**	**Line Category ↑**	**Female vs. Male ↑**	**Manual Behaviors ↑**
Trajectory-Perceiver	.394	.418	.513	.573	.982	.197
Trajectory-GPT	.363	.515	.500	.557	.873	.246

Table 6.5: Benchmarking the community-contributed methods. .The best community-contributed methods perform on par or better with self-supervised video representation learning methods. For mice we also have a trajectory-based method to compare to the video-based methods directly. We find that the trajectory-based method generally does not perform as well as the video-based methods on the mouse dataset. For fly task groups, "Fly type" corresponds to tasks 1 to 11, "Stimulation Control" is tasks 12 to 21, "Stimulation Aggression" is tasks 22 to 36, "Line Category" is tasks 37 to 43, and "Manual Behaviors" is tasks 45 to 50 in "Descrption of fly tasks" in supplementary material of [57]. ↓ indicates MSE and ↑ indicates F1 score. The best-performing model is in bold.

Effect of Masking Strategy

We explore how different masking strategies (spatiotemporal random/cube, temporal/tube and spatial/frame from MAE [17]) affect downstream task performance (Table 6.3), and we use best performing masking ratios used in MAE. We find that contrary to [17, 62], where performances for spatio-temporally agnostic masking (cube) and temporal masking (tube) are very similar to each other, our performance depends on the dataset (mouse or beetle). For the mouse dataset, cube/tube masking have the best overall performance, while for the beetle dataset, cube performs best overall. Overall the differences in performance are also bigger than in [17, 62]. This difference in performance for different masking strategies is likely due to the different spatio-temporal structure of the data, i.e., if the data is more "temporal heavy" or more "spatial heavy."

Effect of ρ on BYOL

We performed ρBYOL [16] with multiple values of ρ, i.e., the number of temporal clips sampled as positives (Table 6.4). In [16], a larger number of ρ steadily increases downstream task performance. This is not true for our datasets, where for mice a value of 2 performs best for 2 tasks and a value of 4 for 2 other tasks. For the beetle dataset, 3 positive samples achieve the best BYOL performance. This is likely to the temporally random sampling of positives for BYOL. This is likely due to the temporally agnostic sampling method for the clips resulting in positives that are of different actions (as the actions of the animals can change rapidly over temporally close frames). Further research is needed on how the temporal sampling strategy for positives needs to be adjusted for temporally heavy datasets.

Transfer Learning from Kinetics400

We evaluated how ρBYOL, Maskfeat, and MAE perform when pre-trained on kinetics400 [33] (Table 6.2). We find that MAE and Maskfeat training on MABe22 generally performs better than using the pre-trained models. Interestingly, for ρBYOL we find the opposite, in that the pre-trained model on Kinetics400 actually performs stronger than counterpart trained on MABe22. Surprisingly, for action recognition, it performed stronger than any of the other models for both mice and beetle data. These results suggest that for action recognition, transfer learning from human datasets to animal datasets is possible to a degree.

Community-Contributed Methods Results

We compare community-contributed methods across all datasets in MABe22 (Table 6.5). The best-performing community methods employ large pre-trained vision models, variations of contrastive learning [11, 25], trajectory data as additional inputs and hand-crafted features (See supplementary material of [57]). Usually, these features are then concatenated and PCA is performed to produce vectors with the embedding dimension. For the mouse dataset, we also compared the top trajectory-based method to the video-based methods on the same data subset. While the performance of the trajectory model for behavior classification is similar to the third-best video-based model, the performance on all other downstream tasks is worse. This is likely due to the loss of visual features after transforming the video frames to sparse keypoint locations. An interesting direction for future work would be to explore how these modalities can be best combined. For the fly dataset (which consists of trajectory data only), we find that using a Perceiver model [30] trained on a masked modeling task works best (See supplementary material of [57]). The second best method is using a GPT [6]-like architecture that generates embeddings from the recurrent trajectory data of all agents. This method is trained using a prediction pretext task.

In general, we find that performance is comparable between community-contributed methods to state-of-the-art video representation learning methods evaluated in Section 6.5. We note that community methods did perform better at learning manual behaviors. This may be due to the hand-crafted features used in the community-contributed methods, which has been shown to be effective at encoding domain knowledge for behavior analysis [58].

Discussion

Overall, we find that methods that perform best on human datasets may not perform the best on our animal datasets. This is likely because human action datasets contain extraneous visual information, whereas our animal datasets minimize these visual cues (consistent backgrounds) and thus behavioral representations need to focus on spatio-temporal information. This highlights a crucial shortcoming of current benchmarks, which may be pushing the community to develop methods that do not focus on the spatio-temporal nature of behavior. We hope to encourage evaluation of representation learning methods on a broader range of settings beyond human videos and annotations, in order to facilitate development of new methods for representation learning and behavior analysis.

References

[1] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. "Learning to See by Moving." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 37–45.

[2] David J. Anderson and Pietro Perona. "Toward a Science of Computational Ethology." In: *Neuron* 84.1 (2014), pp. 18–31.

[3] Gordon J. Berman et al. "Mapping the Stereotyped Behaviour of Freely Moving Fruit Flies." In: *Journal of the Royal Society Interface* 11.99 (2014), p. 20140672.

[4] Brian Thomas Bloomquist et al. "Isolation of a Putative Phospholipase C Gene of Drosophila, norpA, and its role in Phototransduction." In: *Cell* 54.5 (1988), pp. 723–733.

[5] James P. Bohnslav et al. "DeepEthogram, a Machine Learning Pipeline for Supervised Behavior Classification from Raw Pixels." In: *eLife* 10 (2021), e63377.

[6] Tom B. Brown et al. “Language Models Are Few-Shot Learners.” In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1877–1901.

[7] Xavier P. Burgos-Artizzu et al. “Social Behavior Recognition in Continuous Video.” In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 1322–1329.

[8] Adam J. Calhoun, Jonathan W. Pillow, and Mala Murthy. “Unsupervised Identification of the Internal States that Shape Natural Behavior.” In: *Nature neuroscience* 22.12 (2019), pp. 2040–2049.

[9] Mathilde Caron et al. “Unsupervised Learning of Visual Features by Contrasting Cluster Assignments.” In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9912–9924.

[10] Ming-Fang Chang et al. “Argoverse: 3D Tracking and Forecasting with Rich Maps.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8748–8757.

[11] Ting Chen et al. “A Simple Framework for Contrastive Learning of Visual Representations.” In: *International Conference on Machine Learning* (2020).

[12] Navneet Dalal and Bill Triggs. “Histograms of Oriented Gradients for Human Detection.” In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. IEEE. 2005, pp. 886–893.

[13] Tom Decroos, Jan Van Haaren, and Jesse Davis. “Automatic Discovery of Tactics in Spatio-Temporal Soccer Match Data.” In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 223–232.

[14] Jacob Devlin et al. “BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding.” In: *arXiv preprint arXiv:1810.04805* (2018).

[15] Eyrun Eyjolfsdottir et al. “Detecting Social Actions of Fruit Flies.” In: *European Conference on Computer Vision*. Springer. 2014, pp. 772–787.

[16] Christoph Feichtenhofer et al. “A Large-Scale Study on Unsupervised Spatiotemporal Representation Learning.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 3299–3309.

[17] Christoph Feichtenhofer et al. “Masked Autoencoders As Spatiotemporal Learners.” In: *arXiv preprint arXiv:2205.09113* (2022).

[18] Willem E. Frankenhuis, Karthik Panchanathan, and Andrew G. Barto. “Enriching Behavioral Ecology with Reinforcement Learning Methods.” In: *Behavioural Processes* 161 (2019), pp. 94–100.

[19] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. “Unsupervised Representation Learning by Predicting Image Rotations.” In: *ICLR* (2018).

[20] Ross Goroshin et al. “Unsupervised Learning of Spatiotemporally Coherent Metrics.” In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4086–4093.

[21] Jean-Bastien Grill et al. “Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning.” In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21271–21284.

[22] William H Guss et al. “MineRL: A Large-Scale Dataset of Minecraft Demonstrations.” In: *arXiv preprint arXiv:1907.13440* (2019).

[23] Tengda Han, Weidi Xie, and Andrew Zisserman. “Video Representation Learning by Dense Predictive Coding.” In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2019.

[24] Kaiming He et al. “Masked Autoencoders Are Scalable Vision Learners.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16000–16009.

[25] Kaiming He et al. “Momentum Contrast for Unsupervised Visual Representation Learning.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9729–9738.

[26] Dami’an G. Hern’andez et al. “A Framework for Studying Behavioral Evolution by Reconstructing Ancestral Repertoires.” In: *arXiv preprint arXiv:2007.09689* (2020).

[27] Katja Hofmann. “Minecraft as AI Playground and Laboratory.” In: *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*. 2019, pp. 1–1.

[28] Weizhe Hong et al. “Automated Measurement of Mouse Social Behaviors Using Depth Sensing, Video Tracking, and Machine Learning.” In: *Proceedings of the National Academy of Sciences* 112.38 (2015), E5351–E5360.

[29] Alexander I. Hsu and Eric A. Yttri. “B-SOiD: An Open Source Unsupervised Algorithm for Discovery of Spontaneous Behaviors.” In: *bioRxiv* (2020), p. 770271.

[30] Andrew Jaegle et al. “Perceiver IO: A General Architecture for Structured Inputs and Outputs.” In: *arXiv preprint arXiv:2107.14795* (2021).

[31] Mayank Kabra et al. *APT: Animal Part Tracker v0.3.4*. Version v0.3.4. Mar. 2022. DOI: 10.5281/zenodo.6366082. URL: https://doi.org/10.5281/zenodo.6366082.

[32] Mayank Kabra et al. “JAABA: Interactive Machine Learning for Automatic Annotation of Animal Behavior.” In: *Nature Methods* 10.1 (2013), p. 64.

[33] Will Kay et al. “The Kinetics Human Action Video Dataset.” In: *arXiv preprint arXiv:1705.06950* (2017).

[34] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes.” In: *International Conference on Learning Representations*. 2014.

[35] Nathan C. Klapoetke et al. “Independent Optical Excitation of Distinct Neural Populations.” In: *Nature Methods* 11.3 (2014), pp. 338–346.

[36] Ugne Klibaite et al. “An Unsupervised Method for Quantifying the Behavior of Paired Animals.” In: *Physical Biology* 14.1 (2017), p. 015006.

[37] Ugne Klibaite et al. “Deep Phenotyping Reveals Movement Phenotypes in Mouse Neurodevelopmental Models.” In: *Molecular Autism* 13.1 (2022), pp. 1–18.

[38] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. “Revisiting Self-Supervised Visual Representation Learning.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1920–1929.

[39] Hildegard Kuehne et al. “HMDB: A Large Video Database for Human Motion Recognition.” In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 2556–2563.

[40] Yanghao Li et al. “MViTv2: Improved Multiscale Vision Transformers for Classification and Detection.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 4804–4814.

[41] Kevin Luxem et al. “Identifying Behavioral Structure from Deep Variational Embeddings of Animal Motion.” In: *bioRxiv* (2020).

[42] Jeffrey E. Markowitz et al. “The Striatum Organizes 3D Behavior via Moment-to-Moment Action Selection.” In: *Cell* 174.1 (2018), pp. 44–58.

[43] Markus Marks et al. “Deep-Learning-Based Identification, Tracking, Pose Estimation, and Behavior Classification of Interacting Primates and Mice in Complex Environments.” In: *Nature Machine Intelligence* 4.4 (2022), pp. 331–340.

[44] João C. Marques et al. “Structure of the Zebrafish Locomotor Repertoire Revealed with Unsupervised Behavioral Clustering.” In: *Current Biology* 28.2 (2018), pp. 181–195.

[45] Leland McInnes, John Healy, and James Melville. “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.” In: *arXiv preprint arXiv:1802.03426* (2018).

[46] Ishan Misra, C. Lawrence Zitnick, and Martial Hebert. “Shuffle and Learn: Unsupervised Learning Using Temporal Order Verification.” In: *European Conference on Computer Vision*. 2016.

[47] Xun Long Ng et al. “Animal Kingdom: A Large and Diverse Dataset for Animal Behavior Understanding.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 19023–19034.

[48] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation Learning with Contrastive Predictive Coding.” In: *arXiv preprint arXiv:1807.03748* (2018).

[49] Talmo D. Pereira, Joshua W. Shaevitz, and Mala Murthy. “Quantifying Behavior to Understand the Brain.” In: *Nature Neuroscience* 23.12 (2020), pp. 1537–1549.

[50] Alice A. Robie et al. “Mapping the Neural Substrates of Behavior.” In: *Cell* 170.2 (2017), pp. 393–406.

[51] Mikayel Samvelyan et al. “The StarCraft Multi-Agent Challenge.” In: *arXiv preprint arXiv:1902.04043* (2019).

[52] Catherine E. Schretter et al. “Cell Types and Neuronal Circuitry Underlying Female Aggression in Drosophila.” In: *eLife* 9 (2020), e58942.

[53] Martin Schrimpf et al. “Integrative Benchmarking to Advance Neurally Mechanistic Models of Human Intelligence.” In: *Neuron* (2020). URL: `https://www.cell.com/neuron/fulltext/S0896-6273(20)30605-X`.

[54] Cristina Segalin, Jalani Williams, Tomomi Karigo, May Hui, Moriel Zelikowsky, Jennifer J. Sun, Pietro Perona, David J. Anderson, and Ann Kennedy. “The Mouse Action Recognition System (MARS) Software Pipeline for Automated Analysis of Social Behaviors in Mice.” In: *eLife* 10 (2021), e63720.

[55] Keith Sheppard et al. “Stride-level Analysis of Mouse Open Field Behavior using Deep-learning-based Pose Estimation.” In: *Cell Reports* (2022).

[56] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. “UCF101: A Dataset of 101 Human Actions Classes from Videos in the Wild.” In: *arXiv preprint arXiv:1212.0402* (2012).

[57] Jennifer J. Sun, Markus Marks, Andrew Ulmer, Dipam Chakraborty, Brian Geuther, Edward Hayes, Heng Jia, Vivek Kumar, Zachary Partridge, Alice Robie, Catherine E. Schretter, et al. “MABe22: A Multi-Species Multi-Task Benchmark for Learned Representations of Behavior.” In: *International Conference on Machine Learning* (2023). URL: `https://arxiv.org/pdf/2207.10553.pdf`.

[58] Jennifer J. Sun, Ann Kennedy, Eric Zhan, David J. Anderson, Yisong Yue, and Pietro Perona. “Task Programming: Learning Data Efficient Behavior Representations.” In: *Proceedings of the IEEE/CVF Conference on Com-*

puter Vision and Pattern Recognition (CVPR) (2021), pp. 2876–2885. URL: `https://arxiv.org/pdf/2011.13917.pdf`.

[59] Jennifer J. Sun, Tomomi Karigo, Dipam Chakraborty, Sharada Mohanty, Benjamin Wild, Quan Sun, Chen Chen, David Anderson, Pietro Perona, et al. "The Multi-Agent Behavior Dataset: Mouse Dyadic Social Interactions." In: *Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track* (2021).

[60] Ke Sun et al. "Deep High-Resolution Representation Learning for Human Pose Estimation." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5693–5703.

[61] Pei Sun et al. "Scalability in Perception for Autonomous Driving: Waymo Open Dataset." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2446–2454.

[62] Zhan Tong et al. "VideoMAE: Masked Autoencoders Are Data-Efficient Learners for Self-Supervised Video Pre-Training." In: *arXiv preprint arXiv:2203.12602* (2022).

[63] Grant Van Horn et al. "Benchmarking Representation Learning for Natural World Image Collections." In: *Computer Vision and Pattern Recognition*. 2021.

[64] Ashish Vaswani et al. "Attention Is All You Need." In: *Advances in Neural Information Processing Systems* 30 (2017).

[65] Jacob Walker et al. "An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders." In: *European Conference on Computer Vision*. Springer. 2016, pp. 835–851.

[66] Chen Wei et al. "Masked Feature Prediction for Self-Supervised Visual Pre-Training." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 14668–14678.

[67] Alexander B. Wiltschko et al. "Mapping Sub-Second Structure in Mouse Behavior." In: *Neuron* 88.6 (2015), pp. 1121–1135.

[68] Alexander B. Wiltschko et al. "Revealing the Structure of Pharmacobehavioral Space Through Motion Sequencing." In: *Nature Neuroscience* 23.11 (2020), pp. 1433–1443.

[69] Yisong Yue et al. "Learning Fine-Grained Spatial Models for Dynamic Sports Play Prediction." In: *2014 IEEE International Conference on Data Mining*. IEEE. 2014, pp. 670–679.

[70] Eric Zhan*, Jennifer J. Sun*, Ann Kennedy, Yisong Yue, and Swarat Chaudhuri. "Unsupervised Learning of Neurosymbolic Encoders." In: *Transactions on Machine Learning Research* (2022). URL: `https://arxiv.org/pdf/2107.13132.pdf`.

Chapter 7

NEUROSYMBOLIC REPRESENTATIONS

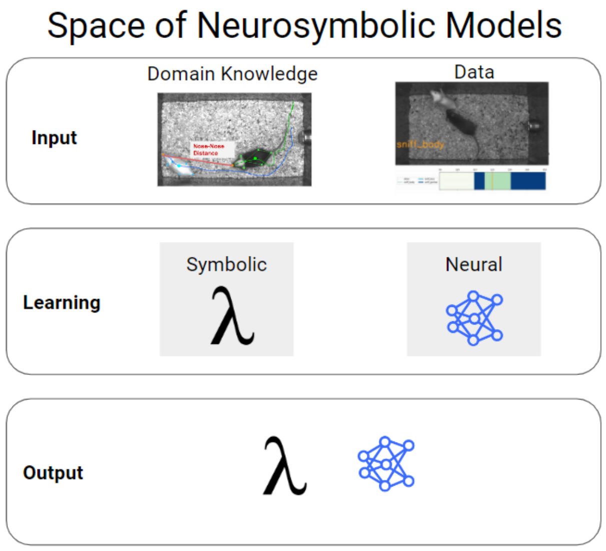


Figure 7.1: Overview of the space of neurosymbolic models.

This chapter is mainly based on the following paper:

[1] Eric Zhan*, Jennifer J. Sun*, Ann Kennedy, Yisong Yue, and Swarat Chaudhuri. “Unsupervised Learning of Neurosymbolic Encoders.” In: *Transactions on Machine Learning Research* (2022). URL: `https://arxiv.org/pdf/2107.13132.pdf`.

Abstract. We present a framework for the unsupervised learning of neurosymbolic encoders, which are encoders obtained by composing neural networks with symbolic programs from a domain-specific language. Our framework naturally incorporates symbolic expert knowledge into the learning process, which leads to more interpretable and factorized latent representations compared to fully neural encoders. We integrate modern program synthesis techniques with the variational autoencoding (VAE) framework, in order to learn a neurosymbolic encoder in conjunction with a standard decoder. The programmatic descriptions from our encoders can benefit many analysis workflows, such as in behavior modeling where interpreting

agent actions and movements is important. We evaluate our method on learning latent representations for real-world trajectory data from animal biology and sports analytics. We show that our approach offers significantly better separation of meaningful categories than standard VAEs and leads to practical gains on downstream analysis tasks, such as for behavior classification.

7.1 Introduction

Advances in unsupervised learning have enabled the discovery of latent structures in data from a variety of domains, such as image data [13], sound recordings [7], and tracking data [28]. For instance, a common approach is to use encoder-decoder frameworks, such as variational autoencoders (VAEs) [24], to identify a low-dimensional latent representation from the raw data that could contain disentangled factors of variation [13] or semantically meaningful clusters [28]. Such approaches typically employ complex mappings based on neural networks, and explaining how the model assigns inputs to latent representations can be challenging [46].

In this paper, we introduce *unsupervised neurosymbolic representation learning*, which allows part of a representation to be computed using symbolic *encoder programs* written in a predefined domain-specific language (DSL). (The rest of the representation is computed using a neural network.) The use of such neurosymbolic encoders can offer two key benefits over purely neural approach. First, since a DSL reflects structured domain knowledge, neurosymbolic encoders can often produce representations that are human-interpretable [42, 36]. Second, as observed in studies that used hand-crafted programmatic encoders [43], these representations can potentially be more factorized or well-separated into meaningful categories than purely neural representations.

Our learning algorithm is grounded in the VAE framework [24, 30] and aims to discover a neurosymbolic encoder coupled with a standard neural decoder.[1] A key challenge here is that the space of programs in a DSL is combinatorial. We tackle this problem by assuming programs to be differentiable and by tightly integrating standard VAE training with modern program synthesis methods [8, 36]. We further show how to incorporate ideas from adversarial information factorization [10] and enforcing capacity constraints [5, 13] in order to mitigate issues such as posterior and index collapse in the learned representation.

[1]Some prior work have studied the complementary problem of learning (neuro-)symbolic decoders (e.g,. [**ellis2017learning**, 15]).

Programmatic descriptions from neurosymbolic encoders are especially useful in behavior analysis [35, 37], where domain experts routinely interpret clusters of behaviors as part of an analysis workflow. Accordingly, our experimental evaluation focuses on this setting. By integrating domain knowledge using program synthesis, we demonstrate that our clusters are inherently interpretable and better aligned with human-annotated labels across multiple behavior analysis datasets. To validate the end-to-end practicality for analysis workflows, we integrate our automatically learned programs into a state-of-the-art behavior analysis framework, Task Programming [37], that typically relies on expert-crafted programs, and demonstrate competitive performance using our automatically synthesized programs.

To summarize, our contributions are:

- We propose a neurosymbolic approach to representation learning, in which part of the latent representation is produced by an interpretable encoder program, while the rest is computed using a neural network.
- We realize the approach via a learning algorithm that combines VAE training and program synthesis.
- We show that our approach can significantly outperform purely neural encoders in extracting semantically meaningful representations of behavior, as measured by standard unsupervised metrics.
- We further explore the flexibility of our approach, by showing that performance can be robust across different DSL designs by domain experts.
- We showcase the practicality of our approach on downstream tasks, by incorporating our approach into a state-of-the-art self-supervised learning approach for behavior analysis [37].

7.2 Background

Variational Autoencoders

We build on VAEs [24, 30], a latent variable modeling framework shown to learn effective latent representations (also called encodings/embeddings) [17, 47, 26] and can capture the generative process [31, 40, 43]. VAEs introduce a latent variable $\mathbf{z}$, an encoder q_ϕ, a decoder p_θ, and a prior distribution p on $\mathbf{z}$. ϕ and θ are the parameters of the q and p, respectively, often instantiated with neural networks.

The learning objective is to maximize the evidence lower bound (ELBO) of the data log-likelihood:

$$\text{ELBO} := \quad \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x}|\mathbf{z})\big] - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big) \leq \quad \log p(\mathbf{x}). \tag{7.1}$$

The first term in Eq. 7.1 is the log-density assigned to the data, while the second term is the KL-divergence between the prior and approximate posterior of $\mathbf{z}$. Latent representations $\mathbf{z}$ are often continuous and modeled with a Gaussian prior, but $\mathbf{z}$ can be modeled to contain discrete dimensions as well [25, 19, 13]. Our experiments are focused on behavioral tracking data in the form of trajectories, and so in practice we utilize a trajectory variant of VAEs [33, 43, 37], described in Section 7.3.

One challenge with VAEs (and deep encoder-decoder models in general) is that while the model is expressive, it is often difficult to interpret what is encoded in the latent representation $\mathbf{z}$. Common approaches include taking traversals in the latent space and visualizing the resulting generations [5], or post-processing the latent variables using techniques such as clustering [28]. Such techniques are post-hoc and thus cannot guide (in an interpretable way) the encoder to be biased towards a family of structures. Some recent work have studied how to impose structure in the form of graphical models or dynamics in the latent space [21, 12], and our work can be thought of as a first step towards imposing structure in the form of symbolic knowledge encoded in a domain specific programming language.

Synthesis of Differentiable Programs

Our approach utilizes recent work on the synthesis of differentiable programs [8, 36, 41], where one learns both the discrete structure of the symbolic program (analogous to the architecture of a neural network) as well as the differentiable parameters within that structure. Our formulation closely follows that of [36]. We use a domain-specific programming language (DSL), generated with a context-free grammar (see Figure 10.2 for an example). A program is represented as a pair (α, ψ), where α is a discrete program architecture and ψ are its real-valued parameters. We denote $\mathcal{P}$ as the space of symbolic programs (i.e., programs with complete architectures). The semantics of a program (α, ψ) is given by a function $[[\alpha]](x, \psi)$ that is guaranteed to be differentiable in both x and ψ.

Like [36], we pose the problem of learning differentiable programs as search through a directed program graph $\mathcal{G}$. The graph $\mathcal{G}$ models the top-down construction of program architectures α through the repeated firing of rules of the DSL grammar, starting with an *empty* architecture α_0 (represented by the "start" nonterminal of the

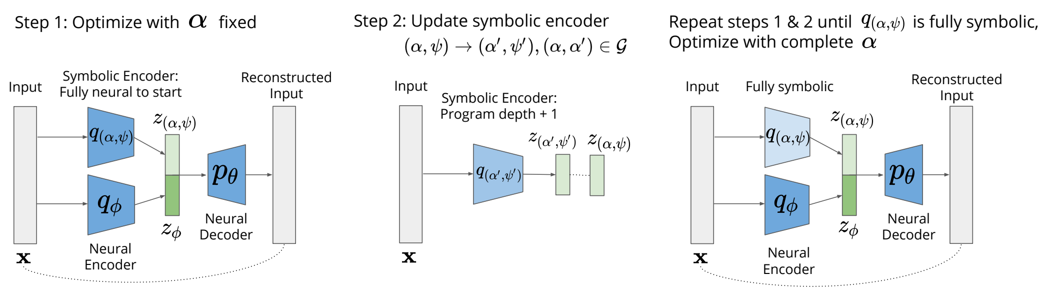


Figure 7.2: Learning Neurosymbolic Encoders: Sketch of Algorithm 1 (Section 7.3). The symbolic encoder is initially fully neural. We alternate between VAE training with the program architecture fixed (Step 1 as in Eq. 7.2), and supervised program learning to increase the depth of the program by 1 (Step 2 as in Eq. 7.3). Once we reach a symbolic program, we train the model one last time to learn all the parameters. The color (in terms of lightness) of the symbolic encoder corresponds to the encoder becoming more symbolic over time.

grammar). The *leaf nodes* of $\mathcal{G}$ represent programs with *complete* architectures (no nonterminals). Thus, $\mathcal{P}$ is the set of programs in the leaf nodes of $\mathcal{G}$. The other nodes in $\mathcal{G}$ contain programs with *partial* architectures (has at least one nonterminal). We interpret a program in a non-leaf node as being neurosymbolic, by viewing its nonterminals as representing neural networks with free parameters. The root node in $\mathcal{G}$ is the empty architecture α_0, interpreted as a fully neural program. An edge (α, α') exists in $\mathcal{G}$ if one can obtain α' from α by applying a rule in the DSL that replaces a nonterminal in α.

Program synthesis in this problem setting equates to searching through $\mathcal{G}$ to find the optimal complete program architecture, and then learning corresponding parameters ψ, i.e., to find the optimal (α, ψ) that minimizes a combination of standard training loss (e.g., classification error) and structural loss (preferring "simpler" α's). [36] evaluate multiple strategies for solving this problem and finds *informed search using admissible neural heuristics* to be the most efficient strategy (see appendix of [44]). Consequently, we adopt this algorithm for our program synthesis task.

7.3 Neurosymbolic Encoders

The structure of our neurosymbolic encoder is shown in the right diagram of Figure 7.2. The latent representation $\mathbf{z} = [\mathbf{z}_\phi, \mathbf{z}_{(\alpha,\psi)}]$ is partitioned into neurally encoded $\mathbf{z}_\phi$ and programmatically encoded $\mathbf{z}_{(\alpha,\psi)}$. This approach boasts several advantages:

- The symbolic component of the latent representation is programmatically interpretable.

- The neural component can encode any residual information not captured by the program, which maintains the model's capacity compared to deep encoders (see synthetic experiment in Section 7.4).
- By incorporating a modular design, we can leverage state-of-the-art learning algorithms for both differentiable encoder-decoder training and program synthesis.

We denote q_ϕ and $q_{(\alpha,\psi)}$ as the neural and symbolic encoders, respectively (see Figure 7.2), where $\mathbf{z}_\phi \sim q_\phi(\cdot|\mathbf{x})$ and $\mathbf{z}_{(\alpha,\psi)} \sim q_{(\alpha,\psi)}(\cdot|\mathbf{x})$. q_ϕ is instantiated with a neural network, but $q_{(\alpha,\psi)}$ is a differentiable program with architecture α and parameters ψ in some program space $\mathcal{P}$ defined by a DSL. Given an unlabeled training set of $\mathbf{x}$'s, our neurosymbolic-VAE (ns-vae) learning objective becomes:

$$\begin{aligned}
&\max_{\phi,(\alpha,\psi),\theta} \quad \mathcal{L}_{\text{ns-vae}}(\phi,\alpha,\psi,\theta) \\
= &\max_{\phi,(\alpha,\psi),\theta} \quad \mathop{\mathbb{E}}_{q_\phi(\mathbf{z}_\phi|\mathbf{x})q_{(\alpha,\psi)}(\mathbf{z}_{(\alpha,\psi)}|\mathbf{x})} \Big[\underbrace{\log p_\theta(\mathbf{x}|\mathbf{z}_\phi,\mathbf{z}_{(\alpha,\psi)})}_{\text{reconstruction loss}} \Big] \\
&\qquad - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}_\phi|\mathbf{x})||p(\mathbf{z}_\phi)\big)}_{\text{regularization for neural latent}} - \underbrace{D_{KL}\big(q_{(\alpha,\psi)}(\mathbf{z}_{(\alpha,\psi)}|\mathbf{x})||p(\mathbf{z}_{(\alpha,\psi)})\big)}_{\text{regularization for symbolic latent}} .
\end{aligned} \tag{7.2}$$

Compared to the standard VAE objective in Eq. 7.1 for a single neural encoder, Eq. 7.2 has separate KL-divergence terms for the neural and programmatic encoders.

Learning Algorithm

The challenge with solving for Eq. 7.2 is that while (ϕ,ψ,θ) can be optimized via back-propagation with α fixed, optimizing for α is a discrete optimization problem. Since it is difficult to jointly optimize over both continuous and discrete spaces, we take an iterative, alternating optimization approach. We start with a fully neural program (one with empty architecture α_0 as described in Section 7.2) trained using standard differentiable optimization (Figure 7.2, Step 1). We then gradually make it more symbolic (Figure 7.2, Step 2) by finding a program that is a child of the current program in $\mathcal{G}$ (more symbolic by construction of $\mathcal{G}$) that outputs as similar to the current latent representations as possible:

$$\min_{\alpha':(\alpha,\alpha')\in\mathcal{G},\ \psi'} \quad \mathcal{L}_{\text{supervised}}\big(q_{(\alpha,\psi)}(\mathbf{x}), q_{(\alpha',\psi')}(\mathbf{x})\big), \tag{7.3}$$

which can be viewed as a form of distillation (from less symbolic to more symbolic programs) via matching the input/output behavior. We solve Eq. 7.3 by enumerating over all child programs of the current search tree and selecting the best one, which is similar to one iteration of iteratively-deepened depth-first search in [36] (more details in Section 7.3). We alternate between optimizing Eq. 7.2 and Eq. 7.3 until we obtain a complete program. Algorithm 1 outlines this procedure and is guaranteed to terminate if $\mathcal{G}$ is finite by specifying a maximum program depth.

We chose this optimization procedure for two reasons. First, it maximally leverages state-of-the-art tools in both differentiable latent variable modeling (VAE-style training) and supervised program synthesis (for distillation), leading to tractable algorithm design. Second, this procedure never makes a drastic change to the program architecture, leading to relatively stable learning behavior across iterations.

Algorithm 1 Learning a neurosymbolic encoder

```
1: Input: program space $\mathcal{P}$, program graph $\mathcal{G}$
2: initialize $\phi, \psi, \theta, \alpha = \alpha_0$ (empty architecture)
3: while $\alpha$ is not complete do
4:     $\phi, \psi, \theta \leftarrow$ optimize Eq. 7.2 with $\alpha$ fixed
5:     $(\alpha, \psi) \leftarrow$ optimize Eq. 7.3
6: $\phi, \psi, \theta \leftarrow$ optimize Eq. 7.2 with complete $\alpha$
7: Return: encoder $\{q_\phi, q_{(\alpha,\psi)}\}$
```

Algorithm 2 Learning a neurosymbolic encoder with k programs

```
1: Input: program space $\mathcal{P}$, program graph $\mathcal{G}$, $k$
2: for $i = 1..k$ do
3:     fix programs $\{q_{(\alpha_1,\psi_1)}, \ldots, q_{(\alpha_{i-1},\psi_{i-1})}\}$
4:     execute Algorithm 1 to learn $q_{(\alpha_i,\psi_i)}$
5:     remove $q_{(\alpha_i,\psi_i)}$ from $\mathcal{P}$ to avoid redundancies
6: Return: encoder $\{q_\phi, q_{(\alpha_1,\psi_1)}, \ldots, q_{(\alpha_k,\psi_k)}\}$
```

Program Synthesis via NEAR

Our strategy for solving Eq. 7.3 utilizes the setup in [36], also described more in Chapter 10 of this thesis. We summarize the key points below.

Program graph $\mathcal{G}$. [36], described further in Chapter 10 of this thesis, learns programs in a supervised learning setting that minimizes a structural cost s (deeper programs are more costly) and a prediction error ζ:

$$(\alpha^*, \psi^*) = \arg\min_{(\alpha,\psi)}(s(\alpha) + \zeta(\alpha, \psi)). \tag{7.4}$$

[36] construct a program graph $\mathcal{G}$ such that solving Eq. 7.4 equates to finding a leaf node with the minimum path cost on $\mathcal{G}$. Our problem definition in Eq. 7.3 is very similar so we utilize the same program graph. The difference is that our labels are not ground-truth but rather the labels assigned by the current neurosymbolic encoder.

Neural heuristic h. [36] solve Eq. 7.4 by introducing a heuristic as a neural admissible relaxation (NEAR for short). Leveraging a fully differentiable DSL, they use neural networks to fill in for nonterminals in programs and show that the performance of such neurosymbolic programs are underestimates of the total path costs of descendent leaf nodes and thus, can be used as an admissible heuristic. This allows them to integrate their heuristic with several graph search algorithms, of which they adopt A* search and iteratively-deepened depth-first-search with branch-and-bound (IDS-BB). We use IDS-BB in our work.

IDS-BB. The full algorithm for IDS-BB is described in Algorithm 2 in [36]. In our work, this reduces to the following:

1. For the current program, we enumerate its children in $\mathcal{G}$.
2. We compute the heuristic for each child in $\mathcal{G}$ by replacing any nonterminals with neural networks.
3. We commit to the most promising child (with respect to the heuristic) and update the program, which can be viewed as one iteration of the full IDS-BB algorithm.

One key difference is that the original IDS-BB algorithm maintains a frontier ordered by the best heuristics encountered so far. However, our label distributions can change between iterations (since the symbolic component of the encoder is updated and thus, so are the labels it assigns), which invalidates the heuristics computed from previous iterations. This leaves a very interesting direction for future work.

Learning Multiple Programs

The interpretability of latent representations induced by symbolic encoders $q_{(\alpha,\psi)}$ ultimately depends on the DSL. For instance, a program that encodes to one of ten classes may not be very interpretable if it involves a matrix multiplication within the program. Instead, we learn *binary* programs that encode sequences into one of two classes (using binary cross-entropy for $\mathcal{L}_{\text{supervised}}$, a uniform prior on 2-dimensional

$\mathbf{z}_{(\alpha,\psi)}$, and Gumbel-Softmax [20] to sample $\mathbf{z}_{(\alpha,\psi)}$ from the posterior). Figures 7.5a & 7.5b depict learned binary programs that encode mice trajectories and their interpretations.

To encode more than two classes, we simply learn multiple binary programs by extending Eq. 7.2 to sum over $\mathcal{L}_{\text{supervised}}$ for k symbolic programs $\{q_{(\alpha_1,\psi_1)},\ldots,q_{(\alpha_k,\psi_k)}\}$ and corresponding latent representations $\{\mathbf{z}_{(\alpha_1,\psi_1)},\ldots,\mathbf{z}_{(\alpha_k,\psi_k)}\}$. This results in 2^k classes and a solution space that now scales exponentially (e.g., $|\mathcal{P}|^k$). Algorithm 2 outlines our greedy solution that reuses Algorithm 1 by iteratively learning one symbolic program at a time. We leave the exploration of more sophisticated search methods as future work.

Dealing with Posterior and Index Collapse

Deep latent variable models, especially those with discrete latent variables, are notoriously prone to both posterior [3, 9, 31] and index [22] collapse. Since our algorithms optimize for such models repeatedly, they can be susceptible to these failure modes. Below, we summarize two strategies that we found to work well in our setting.[2]

Adversarial information factorization. Index collapse is the phenomenon in which all data is encoded into one class, resulting in a discrete latent variable $\mathbf{z}_{(\alpha,\psi)}$ that is effectively meaningless. Creswell et al. [10] counteracts index collapse by introducing an adversarial network A_ω and maximizing the adversarial loss below to ensure that the adversary A_ω cannot successfully predict $\mathbf{z}_{(\alpha,\psi)}$ from $\mathbf{z}_\phi$.

$$\begin{aligned}
&\max_{\phi,(\alpha,\psi),\theta} \mathcal{L}_{\text{fac}}(\phi,\alpha,\psi,\theta)\\
&= \max_{\phi,(\alpha,\psi),\theta} \mathop{\mathbb{E}}_{q_\phi(\mathbf{z}_\phi|\mathbf{x})q_{(\alpha,\psi)}(\mathbf{z}_{(\alpha,\psi)}|\mathbf{x})} \Big[\log p_\theta(\mathbf{x}|\mathbf{z}_\phi,\mathbf{z}_{(\alpha,\psi)}) + \underbrace{\min_\omega \mathcal{L}_{\text{adv}}\big(A_\omega(\mathbf{z}_\phi),\mathbf{z}_{(\alpha,\psi)}\big)\Big]}_{\text{adversary}}\\
&\qquad - D_{KL}\big(q_\phi(\mathbf{z}_\phi|\mathbf{x})||p(\mathbf{z}_\phi)\big) - D_{KL}\big(q_{(\alpha,\psi)}(\mathbf{z}_{(\alpha,\psi)}|\mathbf{x})||p(\mathbf{z}_{(\alpha,\psi)})\big)
\end{aligned} \tag{7.5}$$

Channel capacity constraint. Posterior collapse is the phenomenon in which the posterior trivially matches the prior exactly (a KL-divergence of 0) but the latent variables are unused by the decoder. [5] and [13] instead force the KL-divergence terms to match capacities C_ϕ and $C_{(\alpha,\psi)}$, which are hyperparameter (see appendix of [44]). Since the KL-divergence is an upper bound on the mutual information

[2]There are many approaches available for tackling both these issues, but we emphasize that these contributions are orthogonal to ours; as techniques for preventing posterior and index collapse improve, so will the robustness of our algorithm.

between latent variables and the data [23, 13], this encourages the latent variables to encode information and aims to prevent posterior collapse.

$$\begin{aligned}
&\max_{\phi,(\alpha,\psi),\theta} \quad \mathcal{L}_{\text{cap}}(\phi,\alpha,\psi,\theta) \\
=& \max_{\phi,(\alpha,\psi),\theta} \quad \mathop{\mathbb{E}}_{q_\phi(\mathbf{z}_\phi|\mathbf{x})q_{(\alpha,\psi)}(\mathbf{z}_{(\alpha,\psi)}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}_\phi, \mathbf{z}_{(\alpha,\psi)})\right] \\
&\quad - \gamma_\phi |D_{KL}(q_\phi(\mathbf{z}_\phi|\mathbf{x})||p(\mathbf{z}_\phi)) - C_\phi| \\
&\quad - \gamma_{(\alpha,\psi)} |D_{KL}(q_{(\alpha,\psi)}(\mathbf{z}_{(\alpha,\psi)}|\mathbf{x})||p(\mathbf{z}_{(\alpha,\psi)})) - C_{(\alpha,\psi)}|
\end{aligned} \tag{7.6}$$

In our algorithms, we augment our initial objective in Eq. 7.2 with Eq. 7.5 and Eq. 7.6:

$$\max_{\phi,(\alpha,\psi),\theta} \mathcal{L}_{\text{ns-vae}}(\phi,\alpha,\psi,\theta) + \lambda_{\text{fac}}\mathcal{L}_{\text{fac}}(\phi,\alpha,\psi,\theta) + \lambda_{\text{cap}}\mathcal{L}_{\text{cap}}(\phi,\alpha,\psi,\theta), \tag{7.7}$$

where $\lambda_{\text{fac}} = \lambda_{\text{cap}} = 1$ in our experiments.

Instantiation for Sequential Domains

The objective in Eq. 7.2 describes a general problem that is applicable to any domain. In our experiments, we focus on sequential trajectory data. Trajectory data is often used in scientific applications where interpretability is desirable, such as behavior discovery [28, 18]. The ability to easily explain the learned latent representation using programs can help domain experts better understand the structure in their data. Trajectory data is also often relatively low dimensional, which helps experts encode domain knowledge into the DSL more easily [36, 37, 43].

In this domain, $\mathbf{x}$ is a trajectory of length T: $\mathbf{x} = \{x_1, \ldots, x_T\}$. We then factorize the log-density in Eq. 7.2 as a product of conditional probabilities:

$$\log p_\theta(\mathbf{x}|\mathbf{z}_\phi, \mathbf{z}_{(\alpha,\psi)}) = \sum_{t=1}^{T} \log p_\theta(x_t|x_{<t}, \mathbf{z}_\phi, \mathbf{z}_{(\alpha,\psi)}). \tag{7.8}$$

When q_ϕ and p_θ are instantiated with recurrent neural networks (RNN), the model is more commonly known as a trajectory-VAE (TVAE) [33].

As the symbolic encoder $q_{(\alpha,\psi)}$ maps sequences to vectors, we adopt a DSL (Figure 10.2) previously used for sequence classification [36]. Our DSL is purely functional and contains both basic algebraic operations and parameterized library functions. Domain experts can easily augment the DSL with their own functions, such as selection functions that select subsets of features that they deem potentially important. We ensure that all programs in our DSL are differentiable by utilizing a smooth approximation of the **if-then-else** construct [36]. Figures 7.5a & 7.5b depict example programs.

$$\begin{aligned} \alpha \;::=\;& x \mid \oplus(\alpha_1,\ldots,\alpha_k) \mid \oplus_\theta(\alpha_1,\ldots,\alpha_k) \\ & \textbf{if } \alpha_1 \textbf{ then } \alpha_2 \textbf{ else } \alpha_3 \mid \textbf{sel}_S\, x \mid \textbf{mapaverage } (\textbf{fun } x_1.\alpha_1)\; x \end{aligned}$$

Figure 7.3: Our DSL for sequential domains, similar to the one used in [36]. x, $\oplus$, and $\oplus_\theta$ represent inputs, basic algebraic operations, and parameterized library functions, respectively. **fun** $x.e(x)$ represents a function that evaluates an expression $e(x)$ over the input x. **sel**$_S$ selects a subset S of the dimensions of the input x. **mapaverage** $g\ x$ applies the function g to every element of the sequence x and returns the average of the results. We employ a differentiable approximation of the **if-then-else** construct.

7.4 Experiments

We take a multi-faceted approach to evaluate our unsupervised learning approach using synthetic data and real-world data from animal behavior and sports analytics. We also show the end-to-end practicality of our programs by applying them to a downstream behavior classification framework. Our research questions are:

- **Q1: Are the clusters created with our programs meaningful?** (Section 7.4). We evaluate this aspect both qualitatively and quantitatively by comparing with the truth generative process on synthetic datasets, as well as by comparing to human annotated labels on real-world datasets.
- **Q2: How sensitive is our approach to different DSL choices?** (Section 7.4). We compare programs learned in our framework from three different DSLs designed by three domain experts for studying animal behavior. The three DSLs (DSL 1, DSL 2, DSL 3) mainly differ in the behavioral features chosen by experts, and are described in the appendix of [44].
- **Q3: Are the programs useful for downstream tasks?** (Section 7.4). Ultimately, the practicality of these methods must be validated by their usefulness in downstream tasks such as those used in scientific analyses. We apply our unsupervised programs to a behavior classification framework called task programming [37]. This framework uses hand-crafted programs for self-supervision, which we replace with our automatically learned programs.

Experimental Setup

Datasets

We summarize the datasets used in our experiments, and provide full details in the appendix of [44].

Synthetic. We generate synthetic trajectories by sampling initial positions and velocities from a Gaussian distribution and introducing 2 ground-truth factors of variation as large external forces in the positive/negative x/y directions that affect velocity, totaling to 4 discrete classes. Velocities are sampled and fixed for the entire trajectory, but we also sample small Gaussian noise at each timestep. We generate 10k/2k/2k trajectories of length 25 for train/validation/test. Figure 7.4a shows 50 trajectories from the training set. This dataset is useful because we can evaluate whether our algorithm can learn programs that match the ground-truth factors of variation (such ground-truth information is not available in real-world datasets).

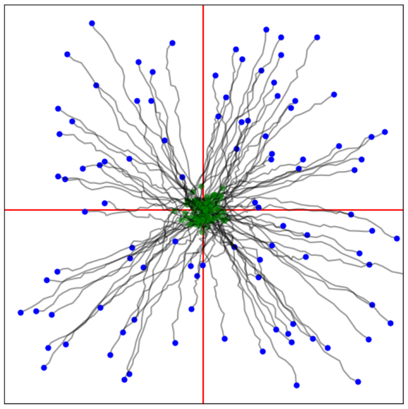
CalMS21. Our primary real-world dataset is the CalMS21 dataset [38], containing trajectories of socially interacting mice captured for neuroscience experiments. Each frame contains 7 tracked keypoints for each of two mice. The dataset has one set of unlabeled tracking data, which we use to train our neurosymbolic encoder, and another set annotated with 4 labels at each frame by human experts (frame-level behaviors), which we use to evaluate our programs. These labels consists of three behaviors-of-interest between mice (attack, mount, investigation), and a label corresponding to all other behaviors (other), with a more detailed description in [38]. Specifically, our evaluation uses labels from the test split of the CalMS21 classification task. We have 231k/52k/262k trajectories of length 21 for train/val/test. The features in our DSL are selected by a domain expert based on the attributes from Segalin et al. [35].

Basketball. We use the same basketball dataset as in [36] and [43] that tracks professional basketball players. Each trajectory is of length 25 over 8 seconds and contains the xy-positions of 10 players. We split trajectories by grouping offensive and defensive players (5 each), effectively doubling the dataset size. We evaluate our algorithm and the baselines with respect to the labels of offensive/defensive players. Our DSL includes additional domain features like player speed and distance-to-basket. In total, we have 177k/31k/27k trajectories for train/val/test.

Quantitative Evaluation Setup

The quantitative evaluations are used to compare our neurosymbolic encoders with baseline unsupervised learning methods on the real-world datasets.

Baselines. We compare our model containing a neurosymbolic encoder against other approaches based on VAEs. In particular, we compare against JointVAE [13], which also has both discrete and continuous latent representations, and can be

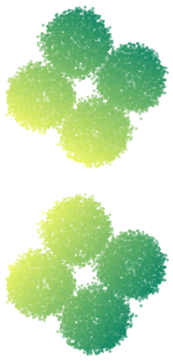


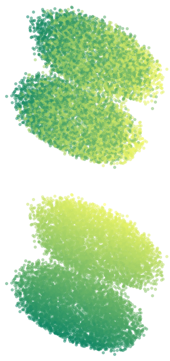
(a) 50 synthetic trajectories

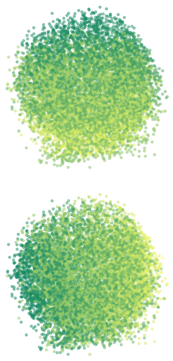
$\mathbb{1}_{[>6.34]}[$
$\quad \mathbf{sel}_{FinalXPosition}\ x]$

$\mathbb{1}_{[>8.99]}[$
$\quad \mathbf{sel}_{FinalYPosition}\ x]$

(b) learned programs







(c) $\mathbf{z}_\phi$, 0 programs (d) $\mathbf{z}_\phi$, 1 program (e) $\mathbf{z}_\phi$, 2 programs

Figure 7.4: Synthetic dataset experiments. **(a)** Trajectories in synthetic training set. Initial/final positions are indicated in green/blue. Red lines delineate ground-truth classes, based on final positions. **(b)** $k = 2$ learned binary programs using our algorithm. The first program (top) thresholds the final x-position while the second program (bottom) thresholds the final y-position. **(c, d, e)** Neural latent variables reduced to 2 dimensions. Top/bottom rows are colored by final x/y-positions, respectively (green/yellow is positive/negative). **(c)** Clusters in TVAE neural latent space correspond to 4 ground-truth classes. **(d)** After learning the first program, the neural latent space contains clusters only based on the final y-position. **(e)** After learning the second program, all 4 ground-truth classes have been extracted as programs and the remaining neural latent space contains no clear clustering.

viewed as a fully neural version of our neurosymbolic encoder. Other baselines include VAE, VAE with K-means loss [29, 28], and Beta-VAE [5]. These models have a fully neural encoder and learn continuous latent representations, which we can then use to produce clusters with K-means clustering [27]. We additionally compare against VQ-VAEs [31], which produce discrete latent clusters. We use the TVAE version of all baselines (details included in the appendix of [44]).

$$1_{[>-7]}\left[\begin{array}{l}\textbf{mapaverage}\ (\textbf{fun}\ x_t.\\ \quad \textbf{multiply}\ (\mathit{ResidentSpeedAffine}_{[-6.3];-8.3}(x_t),\\ \quad\quad \mathit{NoseTailDistAffine}_{[.04];-9.1}(x_t))\ x\end{array}\right]$$

(a) **Program learned using CalMS21 DSL 1**, resulting NMI 0.428. Since speed is positive, the first term is always negative. One cluster thus generally consists of trajectories where the mice are further apart, such that the second term is positive, and the negative product is less than the threshold. The other cluster generally occurs when the mice are close together, the second term is negative, and the product will be positive.

$$1_{[>-5.7]}\left[\begin{array}{l}\textbf{mapaverage}\ (\textbf{fun}\ x_t.\\ \quad \textbf{add}\ (\mathit{ResidentAxisRatioAffine}_{[-8.0];-7.1}(x_t),\\ \quad\quad \mathit{BoundingBoxIOUAffine}_{[-16.6];5.9}(x_t))\ x\end{array}\right]$$

(b) **Program learned using CalMS21 DSL 2**, resulting NMI 0.320. The axis ratio is the ratio of major axis length and minor axis length of an ellipse fitted to the mouse keypoints. The second term measures the bounding box overlap between mice, and is zero when the mice are far apart. It follows that one cluster generally contains trajectories when the mice has larger bounding box overlaps or if the resident axis ratio is large. The other cluster thus contains trajectories where the mice bounding boxes do not overlap, and resident body is compact.

Figure 7.5: Learned programs on CalMS21. The subscripts represents the learned weights (in brackets) and biases (after the brackets) for the affine transformation followed by the bias.

Metrics. Unlike in the synthetic setting, we do not have ground truth programs in the real-world datasets. We thus evaluate our programs quantitatively using (1) standard cluster metrics relative to human-defined labels, and (2) average precision for behavior classification when integrating our programs into downstream tasks. For cluster metrics, we use Purity [34], Normalized Mutual Information (NMI) [45], and Rand Index (RI) [32]. We report the median of three runs. More details, including the standard deviation and the ELBO, are in the appendix.

Q1: Are the clusters created with our programs meaningful?

Synthetic dataset experiments. Our synthetic dataset consists of trajectory data with 4 ground truth classes, corresponding to positive/negative x/y directions. The goal is to learn symbolic programs that capture the ground-truth classes, while leaving the neural latent space to capture any residual information, such as the random initial velocity. We visualize the 2 dimensions of the neural latent space of a TVAE along with 0, 1, and 2 learned programs in Figures 7.4c, 7.4d & 7.4e. The initial neural latent space of the TVAE contains 4 clusters corresponding to the 4 ground-truth classes in Figure 7.4c. After our algorithm learns the first program that

Model	CalMS21			Basketball		
	Purity	NMI	RI	Purity	NMI	RI
Random assignment	.597	.000	.536	.500	.000	.500
TVAE	.598	.089	.564	.501	.001	.500
TVAE+KMeans loss	.605	.118	.573	.501	.001	.500
JointVAE	.597	.019	.537	.560	**.034**	.507
VQ-TVAE	.601	.124	.588	.572	.016	.511
Beta-TVAE	.616	.115	.589	.565	.013	.509
Ours (1 program)	.706	**.423**	**.694**	**.596**	.027	**.518**
Ours (2 programs)	.725	.320	.648	.561	.033	.507
Ours (3 programs)	**.756**	.314	.633	.584	.022	.514

Table 7.1: Evaluating clusters from baseline and our neurosymbolic encoders on human-annotated labels. Median purity, NMI, and RI on CalMS21 and Basketball compared to human-annotated labels (3 runs). Experiment hyperparameters are included in the appendix.

thresholds the final x-position, the resulting latent space in Figure 7.4d captures the other factor of variation as 2 clusters corresponding to the final y-positions. Lastly, when our algorithm learns a second program that thresholds the final y-position, the resulting latent space in Figure 7.4e no longer contains any clear clustering, showing that our approach has successfully extracted the 4 ground-truth classes.

Real-world datasets experiments. We compare clusters produced by our neurosymbolic encoder with fully neural autoencoding baselines (Table 7.1), measured against human-annotated behaviors. For CalMS21, we observe that our method consistently outperforms the baselines in all three cluster metrics. We note that purity increases as the number of programs (thus clusters) increase, while NMI and RI decrease. This implies our method with two clusters best correspond to CalMS21 behaviors, but the other clusters found by our method may still be useful for domain experts. For Basketball, our method improves slightly with respect to purity, but is overall comparable with the baselines.

Qualitative interpretation of our clusters. We further study the programs and clusters produced by our algorithm for the CalMS21 dataset, through a qualitative study with a behavioral neuroscientist. Here, the behavioral neuroscientist analyzes the programmatic clusters produced from the symbolic representation of our neurosymbolic encoder for one, two, and three programs, resulting in two, four, and eight clusters, respectively. The CalMS21 dataset is originally manually annotated with 4 classes corresponding to “attack,” “investigation” (sniff), “mount,” and “other” labels. “Other” corresponds to when no behaviors-of-interest is occurring,

Model	CalMS21 (DSL 1)			CalMS21 (DSL 2)			CalMS21 (DSL 3)		
	Purity	NMI	RI	Purity	NMI	RI	Purity	NMI	RI
Ours (1 program)	.706	**.423**	**.694**	.689	**.364**	**.681**	.649	**.325**	.616
Ours (2 programs)	**.725**	.320	.648	**.715**	.359	.673	**.664**	.324	**.634**

Table 7.2: Effect of varying DSLs on CalMS21 for neurosymbolic encoders. Median purity, NMI, and RI on CalMS21 of our algorithms with DSLs selected by three domain experts compared to human-annotated labels (3 runs). DSL1 corresponds to Table 7.1.

and is typically when the mice are not interacting.

In the single program case, our programs correspond to two discovered clusters. These clusters were classified by domain experts as referring to (1) when the mice are interacting and (2) when there are no interactions. They noted that this is based on distance between the mice, which is consistent with our program (Figure 7.5a) using distance between nose of resident and tail of intruder. For two programs, there are a total of four discovered clusters, with two clusters each corresponding to no interaction and interaction. For the interaction clusters, the domain expert was further able to identify sniff tail behavior as one of the clusters. In this case, the programs found were based on intruder head body angle, resident nose and intruder tail distance, and resident nose and intruder nose distance. The domain expert found the three program case to be more difficult to interpret, but was able to identify clusters corresponding to sniff tail, resident exploration, interaction facing the same direction (for example, mounting), and interaction facing opposite directions (for example, face-to-face sniffing).

Q2: How sensitive is our approach to the DSL?

Choice of DSL. To study the effect of DSL choices, we worked with three domain experts to construct three different DSLs used to learn our programmatic representations. These DSLs contained 8 to 10 different behavioral features for studying mouse social behavior on CalMS21, in addition to common sequential operations (Figure 10.2). A full list of features selected by domain experts are in the appendix.

While there is some variability, our approach consistently outperforms the baselines that contain fully neural encoders for all three DSLs (Table 7.2). Comparing some learned programs from two DSLs (Figures 7.5a, 7.5b), both contain a term that correlates with whether the mice are interacting (distance and bounding box overlap), and another term that correlates with resident speed (mice tends to be more stretched

Model	**CalMS21**			**Basketball**		
	Purity	NMI	RI	Purity	NMI	RI
TVAE	.598	.089	.564	.501	.001	.500
TVAE (w/ features)	.597	.103	.570	.565	.012	.508
VQ-TVAE	.601	.124	.588	.571	.016	.511
VQ-TVAE (w/ features)	.608	.114	.601	.525	.002	.501
Beta-TVAE	.616	.115	.589	.566	.013	.509
Beta-TVAE (w/ features)	.612	.096	.571	.563	.011	.508

Table 7.3: Effect of encoding DSL features into baselines. Median purity, NMI, and RI on CalMS21 and Basketball compared to human-annotated labels (3 runs) for baseline with trajectory inputs only, and baseline with trajectory features added.

when they are moving quickly).

DSL features as input. Lastly, we experiment with using the same DSL features introduced by domain experts as additional features for input trajectories instead (Table 7.3). For both CalMS21 and Basketball, the baselines using the additional features have comparable performance to using input trajectory data alone. In contrast, by using the features more explicitly as part of the DSL in our neurosymbolic encoders, we are able to produce clusters with a better separation between behavior classes based on cluster metrics (see Table 7.1).

Q3: Are the programs useful for downstream tasks?

We apply our programs to frame-level behavior classification [35, 14, 6], where the goal is to automatically quantify behavior based on expert annotations. We are motivated by the observation that manual behavior annotation is time-consuming and expensive [2], often being a bottleneck in the analysis workflow. Our unsupervised programs have the potential to reduce annotation effort and help accelerate behavioral studies, through the task programming framework. Task programming [37] uses hand-crafted programs as self-supervision to improve behavior classification data efficiency; however, hand designing programs still requires human effort. Here, we show that unsupervised programs learned using our neurosymbolic encoders performs comparably to expert-designed programs on CalMS21.

We integrate the learned programs from our neurosymbolic encoder into the task programming framework (i.e., use them as a source of self-supervision instead of the expert-crafted programs), and compare to the classification performance using expert programs (Figure 7.6). The classification performance is computed using

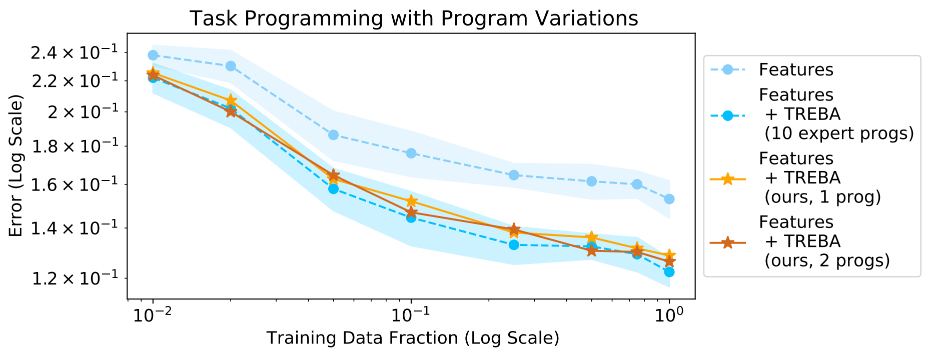


Figure 7.6: Applying symbolic encoders for self-supervision. “Features” is baseline w/o self-supervision. “TREBA” is a self-supervised approach in the Task Programming paradigm [37], using either expert-crafted programs or our symbolic encoders as the weak-supervision rules. The shaded region is std dev over 9 repeats. The std dev for our approach (not shown) is comparable. Based on [37], the error is computed using 1.0 – Mean Average Precision.

Mean Average Precision on the behaviors-of-interest in CalMS21 (attack, investigation, mount). Using only one program found using our approach, we are able to achieve comparable performance to 10 expert-written programs on the behavior classification task studied in [37]. Importantly, we note that we automatically learned the self-supervision tasks from a DSL, instead of hand-crafting them as in [37]. This demonstrates that programs found by our approach can be applied effectively to downstream behavior analysis tasks such as task programming.

7.5 Discussion

We present a novel approach for unsupervised learning of neurosymbolic encoders. Our approach integrates the VAE framework with program synthesis and results in a learned representation with both neural and symbolic components. Experiments on trajectory data from behavior analysis demonstrate that our programmatic descriptions of the latent space result in more meaningful clusters relative to human-defined behaviors, compared to purely neural encoders. Additionally, we show the practicality of our approach by applying our learned programs to achieve comparable performance to expert-constructed tasks in a self-supervised learning approach for behavior classification.

Problem Scope. We explore unsupervised learning of neurosymbolic encoders for the first time, and here, our neurosymbolic encoders tackle domains consisting of lower dimensional spatiotemporal data. These types of domains covers a wide range of application areas, from behavioral data (animal behavior and sports analytics in our experiments), to control systems for rigid-body systems, to biomarkers or socioeconomic markers. In many of these domains, there are existing domain expertise that can be leveraged to create the DSL for our neurosymbolic encoders. For example, we use the behavioral features from [35] in our work. One direct application of learning semantically meaningful programs is that it can be used to improve learning pipelines, such as task programming, as we have demonstrated.

Limitations. One limitation of our current approach is scalability of the program search process. While our program search is parallelizable, such that learning additional programs would not incur significant additional time, the symbolic encoder update does increase the runtime over a purely neural solution. Here, we have explored our approach on settings where shorter programs are beneficial. Future work have the potential to further expand the applications of these models to larger, more complex systems. Furthermore, our approach requires programs that are differentiable with respect to its parameters. We note that there are increasingly more differentiable DSLs, such as [36, 11, 41, 16, 4], and there are commonly-adopted ways to make differentiable approximations to more established non-differentiable DSLs (for example, in [36], the authors use a smooth differentiable approximation of the non-differentiable if-then-else statement). These common challenges in using neurosymbolic learning in science is further discussed in [39] in Chapter 12 of this thesis.

Future Directions. There are many future directions to explore for neurosymbolic encoders based on our work. Scalability is one important area as discussed above. Another direction is to extend this work to other domains such as image and text data, in order to learn interpretable symbolic latent representations. Neurosymbolic encoders on images would require a DSL for pixel data as well as architecture changes, such as using convolutional VAEs. Furthermore, one can improve upon our greedy approach in Algorithm 2 for finding the optimal set of symbolic programs, e.g., by performing local coordinate ascent in program space, similar to algorithms for large-scale neighborhood search [1]. Lastly, while practically-oriented extensions of VAEs such as our own have yielded great practical benefit, they often lead to sub-optimal results from a pure likelihood (or ELBO) perspective. One final direction is to rigorously formulate a learning objective from the ground up that formally encapsulates practically-oriented extensions of VAEs.

References

[1] Ravindra K. Ahuja et al. “A Survey of Very Large-Scale Neighborhood Search Techniques.” In: *Discrete Applied Mathematics* 123.1-3 (2002), pp. 75–102.

[2] David J. Anderson and Pietro Perona. “Toward a Science of Computational Ethology.” In: *Neuron* 84.1 (2014), pp. 18–31.

[3] Samuel R. Bowman et al. “Generating Sentences from a Continuous Space.” In: *arXiv preprint arXiv:1511.06349* (2015).

[4] Rudy R. Bunel et al. “Adaptive Neural Compilation.” In: *Advances in Neural Information Processing Systems* 29 (2016).

[5] Christopher P. Burgess et al. “Understanding Disentanglement in β-VAE.” In: *Neural Information Processing Systems Disentanglement Workshop*. 2017.

[6] Xavier P. Burgos-Artizzu et al. “Social Behavior Recognition in Continuous Video.” In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 1322–1329.

[7] Adam J. Calhoun, Jonathan W. Pillow, and Mala Murthy. “Unsupervised Identification of the Internal States that Shape Natural Behavior.” In: *Nature neuroscience* 22.12 (2019), pp. 2040–2049.

[8] Swarat Chaudhuri et al. “Neurosymbolic Programming.” In: *Foundations and Trends® in Programming Languages* 7.3 (2021), pp. 158–243.

[9] Xi Chen et al. “Variational Lossy Autoencoder.” In: *arXiv preprint arXiv:1611.02731* (2016).

[10] Antonia Creswell et al. “Adversarial Information Factorization.” In: *arXiv preprint arXiv:1711.05175* (2017).

[11] Guofeng Cui and He Zhu. “Differentiable Synthesis of Program Architectures.” In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 11123–11135.

[12] Zhiwei Deng et al. “Factorized Variational Autoencoders for Modeling Audience Reactions to Movies.” In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

[13] Emilien Dupont. “Learning Disentangled Joint Continuous and Discrete Representations.” In: *Advances in Neural Information Processing Systems*. 2018.

[14] Eyrun Eyjolfsdottir et al. “Learning Recurrent Representations for Hierarchical Behavior Modeling.” In: *International Conference on Learning Representations* (2017).

[15] Reuben Feinman and Brenden M. Lake. “Learning Task-General Representations with Generative Neuro-Symbolic Modeling.” In: *International Conference on Learning Representations*. 2020.

[16] Alexander L. Gaunt et al. “Terpret: A Probabilistic Programming Language for Program Induction.” In: *arXiv preprint arXiv:1608.04428* (2016).

[17] Irina Higgins et al. “Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework.” In: *International Conference on Learning Representations*. 2016.

[18] Alexander I. Hsu and Eric A. Yttri. “B-SOiD: An Open Source Unsupervised Algorithm for Discovery of Spontaneous Behaviors.” In: *bioRxiv* (2020), p. 770271.

[19] Zhiting Hu et al. “Toward Controlled Generation of Text.” In: *International Conference on Machine Learning*. 2017.

[20] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical Reparameterization with Gumbel-Softmax.” In: *arXiv preprint arXiv:1611.01144* (2017).

[21] Matthew J. Johnson et al. “Composing Graphical Models with Neural Networks for Structured Representations and Fast Inference.” In: *Advances in Neural Information Processing Systems*. 2016.

[22] Lukasz Kaiser et al. “Fast Decoding in Sequence Models Using Discrete Latent Variables.” In: *International Conference on Machine Learning*. 2018.

[23] Hyunjik Kim and Andriy Mnih. “Disentangling by Factorising.” In: *International Conference on Machine Learning*. 2018.

[24] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes.” In: *International Conference on Learning Representations*. 2014.

[25] Diederik P. Kingma et al. “Semi-Supervised Learning with Deep Generative Models.” In: *arXiv preprint arXiv:1406.5298* (2014).

[26] Yingzhen Li and Stephan Mandt. “Disentangled Sequential Autoencoder.” In: *International Conference on Machine Learning*. 2018.

[27] Stuart Lloyd. “Least Squares Quantization in PCM.” In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[28] Kevin Luxem et al. “Identifying Behavioral Structure from Deep Variational Embeddings of Animal Motion.” In: *bioRxiv* (2020).

[29] Qianli Ma et al. “Learning Representations for Time Series Clustering.” In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 3781–3791.

[30] Andriy Mnih and Karol Gregor. “Neural Variational Inference and Learning in Belief Networks.” In: *International Conference on Machine Learning*. 2014.

[31] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. "Neural Discrete Representation Learning." In: *Advances in Neural Information Processing Systems*. 2017.

[32] William M. Rand. "Objective Criteria for the Evaluation of Clustering Methods." In: *Journal of the American Statistical Association* 66.336 (1971), pp. 846–850.

[33] John Co-Reyes et al. "Self-Consistent Trajectory Autoencoder: Hierarchical Reinforcement Learning with Trajectory Embeddings." In: *International Conference on Machine Learning*. 2018.

[34] Hinrich Schütze, Christopher D. Manning, and Prabhakar Raghavan. *Introduction to Information Retrieval*. Vol. 39. Cambridge University Press Cambridge, 2008.

[35] Cristina Segalin, Jalani Williams, Tomomi Karigo, May Hui, Moriel Zelikowsky, Jennifer J. Sun, Pietro Perona, David J. Anderson, and Ann Kennedy. "The Mouse Action Recognition System (MARS) Software Pipeline for Automated Analysis of Social Behaviors in Mice." In: *eLife* 10 (2021), e63720.

[36] Ameesh Shah*, Eric Zhan*, Jennifer J. Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. "Learning Differentiable Programs with Admissible Neural Heuristics." In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 4940–4952. URL: `https://arxiv.org/pdf/2007.12101.pdf`.

[37] Jennifer J. Sun, Ann Kennedy, Eric Zhan, David J. Anderson, Yisong Yue, and Pietro Perona. "Task Programming: Learning Data Efficient Behavior Representations." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 2876–2885. URL: `https://arxiv.org/pdf/2011.13917.pdf`.

[38] Jennifer J. Sun, Tomomi Karigo, Dipam Chakraborty, Sharada Mohanty, Benjamin Wild, Quan Sun, Chen Chen, David Anderson, Pietro Perona, et al. "The Multi-Agent Behavior Dataset: Mouse Dyadic Social Interactions." In: *Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track* (2021).

[39] Jennifer J. Sun*, Megan Tjandrasuwita*, Atharva Sehgal*, Armando Solar-Lezama, Swarat Chaudhuri, Yisong Yue, and Omar Costilla-Reyes. "Neurosymbolic Programming for Science." In: *AI for Science Workshop at Neural Information Processing Systems (NeurIPS)* (2022). URL: `https://arxiv.org/pdf/2210.05050.pdf`.

[40] Arash Vahdat and Jan Kautz. "Nvae: A Deep Hierarchical Variational Autoencoder." In: *Advances in Neural Information Processing Systems*. 2020.

[41] Lazar Valkov et al. “Houdini: Lifelong Learning as Program Synthesis.” In: *Advances in neural information processing systems*. 2018.

[42] Abhinav Verma et al. “Programmatically interpretable reinforcement learning.” In: *International Conference on Machine Learning*. 2018.

[43] Eric Zhan et al. “Learning Calibratable Policies using Programmatic Style-Consistency.” In: *International Conference on Machine Learning*. 2020.

[44] Eric Zhan*, Jennifer J. Sun*, Ann Kennedy, Yisong Yue, and Swarat Chaudhuri. “Unsupervised Learning of Neurosymbolic Encoders.” In: *Transactions on Machine Learning Research* (2022). URL: `https://arxiv.org/pdf/2107.13132.pdf`.

[45] Hui Zhang et al. “Unsupervised Feature Extraction for Time Series Clustering Using Orthogonal Wavelet Transform.” In: *Informatica* 30.3 (2006).

[46] Yu Zhang et al. “A Survey on Neural Network Interpretability.” In: *arXiv preprint arXiv:2012.14261* (2020).

[47] Shengjia Zhao, Jiaming Song, and Stefano Ermon. “Learning Hierarchical Features from Generative Models.” In: *International Conference on Machine Learning*. 2017.

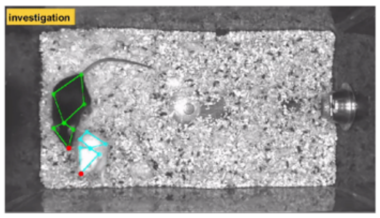
Part III

Integrating Symbolic Domain Knowledge with Learning

Chapter 8

OVERVIEW

The standard supervised learning paradigm requires cleanly pre-defined labels and large amounts of annotations, both of which are difficult or impossible to satisfy for many scientific inquiries. To tackle this, we develop frameworks that learn from scientists to directly incorporate existing domain knowledge into the model (for example, [1]) as well as iteratively improve knowledge formulations from data (for example, [3]). New algorithmic designs are needed to incorporate such structured knowledge. My approach is to develop neurosymbolic learning methods that integrate symbolic knowledge from scientists with the flexibility of neural networks.



```
IF   (distance between noses) < A AND
     (facing angle) < B

THEN investigation IF
     (acceleration of mouse 1) > C

ELSE investigation IF
     (distance from nose 1 to centroid 2) < D
```

Figure 8.1: Example of a neurosymbolic program for mice investigation.

My work combines neural and symbolic components in ways that reveal new properties for neurosymbolic models as well as improve scientific workflows. In particular, we explore the spectrum between:

- Symbolic models: These models typically encode explicit rules or knowledge on the data (for example, a scientist may hand-craft a hypothesis on the data in a general programming language). These models are often used in domains where the underlying knowledge is well-understood, such as logic or physics equations. However, symbolic models can be difficult to scale to complex problems and noisy datasets. Techniques for finding symbolic models include symbolic regression and program synthesis.

- Neurosymbolic models: Neurosymbolic models are models that combine neural networks with symbolic representations. They can learn from data in a way that is similar to neural networks, but encode symbolic domain knowledge. We explore a few ways to combine these modules; from symbolically-guided training of neural networks, to using neural heuristics for more efficient search of neurosymbolic and symbolic architectures.

- Neural models: These models are the most common in driving recent developments in machine learning and deep learning. Neural networks typically have a large set of parameters, very high representation power, and are often used in domains where the underlying knowledge is not well-understood (for example, language modeling or computer vision). However, neural networks can be difficult to interpret and explain. Works in interpretable ML either focuses on post-hoc explanations, or learning inherently interpretable models. Our work is more closely related to the latter.

Towards a general neurosymbolic framework, my work has studied algorithms using neural relaxations to improve the scalability of differentiable program search [1] as well as frameworks that connect symbolic programs and weak supervision to improve data efficiency [3]. In addition, we found that integrating symbolic knowledge also benefits structure discovery, as neurosymbolic models learn more meaningful categories than neural networks alone [4]. I further highlighted opportunities at the intersection of neurosymbolic models and science in a recent perspective paper [2] for the ML community.

The works in this chapter consists of:

- Chapter 9: We guide the training of neural networks using symbolic domain knowledge encoded as programs, to learn more data-efficient representations of behavior.

- Chapter 10: We learn differentiable programs more efficiently, by using neural heuristics to guide the search over the space of symbolic program architectures.

- Chapter 11: We synthesize programs from small amounts of data, and use these programs to automatically supervise neural networks through active learning and weak supervision.

- Chapter 12: We provide a perspective paper on the opportunities and challenges towards a deeper integration of neurosymbolic methods and science, focused on behavior analysis.

References

[1] Ameesh Shah*, Eric Zhan*, Jennifer J. Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. "Learning Differentiable Programs with Admissible Neural Heuristics." In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 4940–4952. URL: https://arxiv.org/pdf/2007.12101.pdf.

[2] Jennifer J. Sun*, Megan Tjandrasuwita*, Atharva Sehgal*, Armando Solar-Lezama, Swarat Chaudhuri, Yisong Yue, and Omar Costilla-Reyes. "Neurosymbolic Programming for Science." In: *AI for Science Workshop at Neural Information Processing Systems (NeurIPS)* (2022). URL: https://arxiv.org/pdf/2210.05050.pdf.

[3] Albert Tseng, Jennifer J. Sun, and Yisong Yue. "Automatic Synthesis of Diverse Weak Supervision Sources for Behavior Analysis." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 2211–2220. URL: https://arxiv.org/pdf/2111.15186.pdf.

[4] Eric Zhan*, Jennifer J. Sun*, Ann Kennedy, Yisong Yue, and Swarat Chaudhuri. "Unsupervised Learning of Neurosymbolic Encoders." In: *Transactions on Machine Learning Research* (2022). URL: https://arxiv.org/pdf/2107.13132.pdf.

Chapter 9

TASK PROGRAMMING

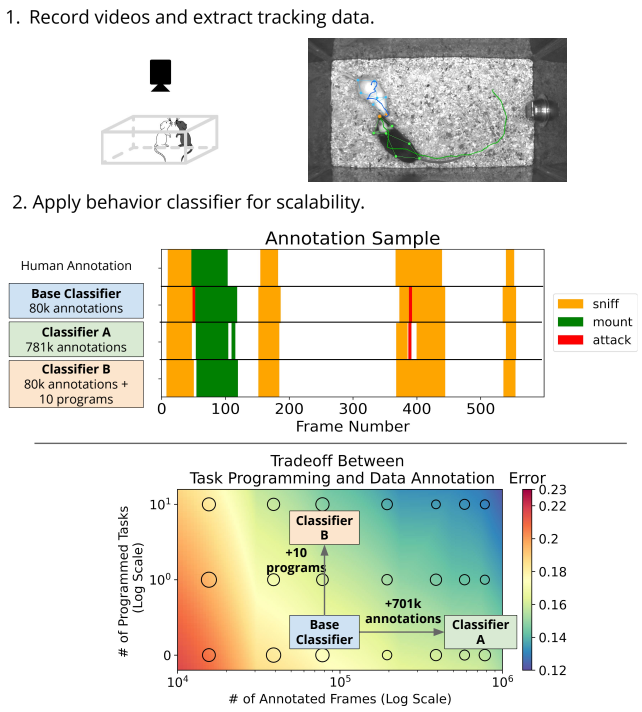


Figure 9.1: Overview of task programming. *Part 1:* A typical behavior study starts with extraction of tracking data from videos. We show 7 keypoints for each mouse, and draw the trajectory of the nose keypoint. *Part 2:* Domain experts can either do data annotation (Classifier A) or task programming (Classifier B) to reduce classifier error. The middle panel shows annotated frames at 30Hz. Colors in the bottom plot represent interpolated performance based on classifier error at the circular markers. The size of the marker represents the error variance.

This chapter is mainly based on the following paper:

[1] Jennifer J. Sun, Ann Kennedy, Eric Zhan, David J. Anderson, Yisong Yue, and Pietro Perona. "Task Programming: Learning Data Efficient Behavior Representations." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 2876–2885. URL: `https://arxiv.org/pdf/2011.13917.pdf`.

Abstract. Specialized domain knowledge is often necessary to accurately annotate training sets for in-depth analysis, but can be burdensome and time-consuming to acquire from domain experts. This issue arises prominently in automated behavior analysis, in which agent movements or actions of interest are detected from video tracking data. To reduce annotation effort, we present TREBA: a method to learn annotation-sample efficient trajectory embedding for behavior analysis, based on multi-task self-supervised learning. The tasks in our method can be efficiently engineered by domain experts through a process we call "task programming", which uses programs to explicitly encode structured knowledge from domain experts. Total domain expert effort can be reduced by exchanging data annotation time for the construction of a small number of programmed tasks. We evaluate this trade-off using data from behavioral neuroscience, in which specialized domain knowledge is used to identify behaviors. We present experimental results in three datasets across two domains: mice and fruit flies. Using embeddings from TREBA, we reduce annotation burden by up to a factor of 10 without compromising accuracy compared to state-of-the-art features. Our results thus suggest that task programming and self-supervision can be an effective way to reduce annotation effort for domain experts.

9.1 Introduction

Behavioral analysis of one or more agents is a core element in diverse fields of research, including biology [36, 25], autonomous driving [6, 40], sports analytics [43, 45], and video games [19, 3]. In a typical experimental workflow, the location and pose of agents is first extracted from each frame of a behavior video, and then labels for experimenter-defined behaviors of interest are applied on a frame-by-frame basis based on the pose and movements of the agents. In addition to reducing human effort, automated quantification of behavior can lead to more objective, precise, and scalable measurements compared to manual annotation [1, 9]. However, training behavior detection models can be data intensive and manual behavior annotation often requires specialized domain knowledge and high-frequency temporal labels. As a result, this process of generating training datasets is time-consuming and effort-intensive for experts. Therefore, methods to reduce annotation effort by domain experts are needed to accelerate behavioral studies.

We study alternative ways for domain experts to improve classifier accuracy beyond simply increasing the sheer volume of annotations. In particular, we propose a framework that unifies: (1) self-supervised representation learning, and (2) encoding explicit structured knowledge on trajectory data using expert-defined programs. Domain experts can construct these programs efficiently because keypoint trajectories in each frame are typically low dimensional, and experts can already hand-design effective features for trajectory data [36, 27]. To best leverage this structured expert knowledge, we develop a framework to learn trajectory representations based on multi-task self-supervised learning, which has not been well-explored for trajectory data.

Our Approach. Our framework, **Tr**ajectory **E**mbedding for **B**ehavior **A**nalysis (TREBA), learns trajectory representations through trajectory generation alongside a set of decoder tasks based on expert-engineered programs. These programs are created by domain experts through a process we call task programming, inspired by the data programming paradigm [32]. Task programming is a process by which domain experts identify trajectory attributes relevant to the behaviors of interest under study, write programs, and apply those programs to inform representation learning (Section 9.3). This flexibility in decoder tasks allows our framework to be applicable to a variety of agents and behaviors studied across diverse fields of research.

Expert Effort Tradeoffs. Since task programming will typically require a domain expert's time, we study the tradeoff between doing task programming and data annotation. We compare behavior classification performance with different amounts of annotated training data and programmed tasks. For example, for the domain illustrated in Figure 9.1, domain experts can reduce error by 13% relative to the base classifier by annotating 701k additional frames, or they can reduce error by 16% by learning a representation using 10 programmed tasks in our framework. Our approach allows experts to trade a large number of annotations for a small number of programmed tasks.

We study our approach across two domains in behavioral neuroscience, namely mouse and fly behavior. We chose this setting because it requires specialized domain knowledge for data annotation, and data efficiency is important for domain experts. Furthermore, decoder tasks in our framework can be efficiently programmed by experts based on simple functions describing trajectory attributes for identifying behaviors of interest. For example, for mouse social behaviors such as attack [36], important behavior attributes include the speed of each mouse and distance between mice. The corresponding task could then be to decode these attributes from the learned representations.

To summarize our contributions:

- We introduce task programming as an efficient way for domain experts to reduce annotation effort and encode structural knowledge. We develop a novel method to learn an annotation-sample efficient trajectory representation using self-supervision and programmatic supervision.
- We study the effect of task programming, data annotation, and different decoder losses on behavior classifier performance.
- We demonstrate these representations on three datasets in two domains, showing that our method can lead to a 10× annotation reduction for mice, and 2× for flies.

9.2 Related Work

Behavior Modeling. Behavior modeling using trajectory data is studied across a variety of fields [25, 6, 40, 43, 19, 3]. In particular, there is an increasing effort to automatically detect and classify behavior from trajectory data [22, 1, 14, 26, 12]. Our experiments are based on behavior classification datasets from behavioral neuroscience [13, 4, 36], a field where specialized domain knowledge is important for identifying behaviors of interest.

The behavior analysis pipeline generally consists of the following steps: (1) tracking the pose of agents, (2) computing pose-based features, and (3) training behavior classifiers [4, 20, 36, 27]. To address step 1, there are many existing pose estimation models [13, 26, 17]. In our work, we leverage two existing pose models, [36] for mice and [13] for flies, to produce trajectory data. In steps 2 and 3 of the typical behavior analysis pipeline, hand-designed trajectory features are computed from the animals' pose, and classifiers are trained to predict behaviors of interest in a fully supervised fashion [4, 20, 13, 36]. Training fully supervised behavior classifiers requires time-consuming annotations by domain experts [1]. Instead, our proposed approach enables domain experts to trade time-consuming annotation work for task programming with representation learning.

Another group of work uses unsupervised methods to discover new motifs and behaviors [21, 42, 2, 25, 5]. Our work focuses on the more common case where domain experts already know what types of actions they would like to study in an experiment. We aim to improve the data-efficiency of learning expert-defined behaviors.

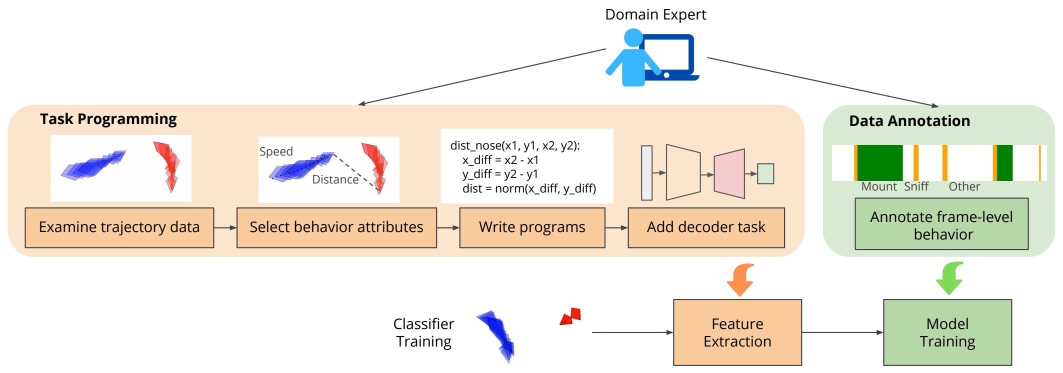


Figure 9.2: Task Programming and Data Annotation for Classifier Training. Domain experts can choose between doing task programming and/or data annotation. Task programming is the process for domain experts to engineer decoder tasks for representation learning. The programs enable learning of annotation-sample efficient trajectory features to improve performance instead of additional annotations.

Representation Learning. Visual representation learning has made great progress in effective representations for images and videos [16, 15, 7, 28, 24, 18, 38]. Self-supervised signals are often used to train this visual representation, such as learning relative positions of image patches [10], predicting image rotations [15], predicting future patches [28], and contrastive learning on augmented images [7]. Compared to visual data, trajectory data is significantly lower dimensional in each frame, and techniques from visual representation learning often cannot be applied directly. For example, while we can create image patches that represent the same visual class, it is difficult to select a partial set of keypoints that represent the same behavior. Our framework builds upon these approaches to learn effective representations for behavioral data.

We investigate different decoder tasks in order to learn an effective behavior representation. One decoder task that we investigate is self-decoding: the reconstruction of input trajectories using generative modeling. Generative modeling has previously been applied to learn representations for visual data [46, 38, 28] and language modeling [30]; for trajectory data, we use imitation learning [41, 44, 45] to train our trajectory representation. The other tasks in our multi-task self-supervised learning framework are created by domain experts using task programming (Section 9.3). This idea of using a human-provided function as part of training has been studied for training set creation [32, 31], and controllable trajectory generation [45]. Our work explores these additional decoder tasks to further improve the learned representation over the generative loss alone.

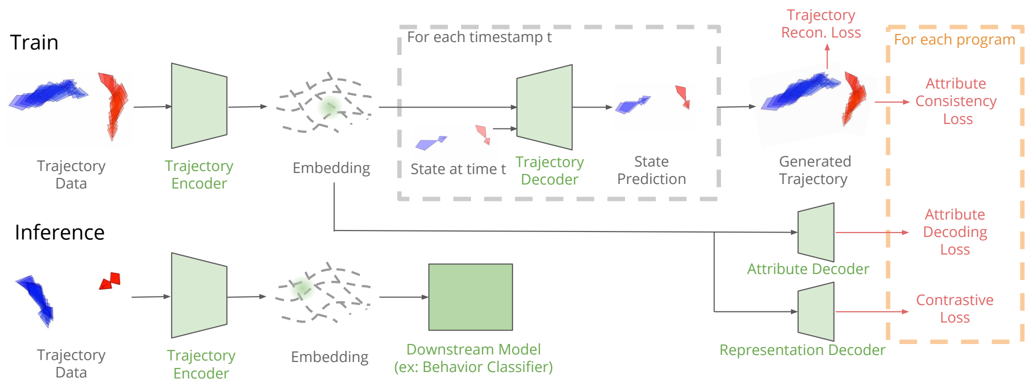


Figure 9.3: TREBA Training and Inference Pipelines. During training, we use trajectory self-decoding and the programmed decoder tasks to train the trajectory encoder. The learned representation is used for downstream tasks such as behavior classification.

Multi-Task Self-Supervised Learning. We jointly optimize a family of self-supervised tasks in an encoder-decoder setup, making this work an example of multi-task self-supervised learning. Multi-task self-supervised learning has been applied to other domains such as visual data [11, 24], accelerometer recordings [35], audio [33] and multi-modal inputs [37, 29]. Generally in each of these domains, tasks are defined ahead of time, as is the case for tasks such as frame reconstruction, colorization, finding relative position of image patches, and video-audio alignment. Most of these tasks are designed for image or video data, and cannot be applied directly to trajectory data. To construct tasks for trajectory representation learning, we propose that domain experts can use task programming to engineer decoder tasks and encode structural knowledge.

9.3 Method

We introduce **Tr**ajectory **E**mbedding for **B**ehavior **A**nalysis (TREBA), a method to learn an annotation-sample efficient trajectory representation using self-supervision and auxiliary decoder tasks engineered by domain experts. Figure 9.2 provides an overview of the expert's role. In our framework, domain experts replace (a significant amount of) time-consuming manual annotation with the construction of a small number of programmed tasks, reducing total expert effort. Each task places an additional constraint on the learned trajectory embedding.

TREBA uses the expert-programmed tasks based on a multi-task self-supervised learning approach, outlined in Figure 9.3. To learn task-relevant low-dimensional representations of pose trajectories, we train a network jointly on (1) reconstruction of the input trajectory (Section 9.3) and (2) expert-programmed decoder tasks (Section 9.3). The learned representation can then be used as input to behavior modeling tasks, such as behavior classification.

Trajectory Representations

Let $\mathcal{D}$ be a set of N unlabelled trajectories. Each trajectory τ is a sequence of states $\tau = \{(s_t)\}_{t=1}^{T}$, where the state s_i at timestep i corresponds to the location or pose of the agents at that timestep. In this study, we divide trajectories from longer recordings into segments of length T, but in general trajectory length can vary. For multiple agents, the keypoints of each agent is stacked at each timestep.

Before we introduce our expert-programmed tasks, we will use trajectory reconstruction as an initial self-supervised task. Given a history of agent states, we would like our model to predict the next state. This task is usually studied with sequential generative models. We used trajectory variational autoencoders (TVAEs) [34, 45] to embed the input trajectory using an RNN encoder, q_ϕ, and an RNN decoder, p_θ, to predict the next state. The TVAE loss is:

$$\mathcal{L}^{\text{tvae}} = \mathbb{E}_{q_\phi}\left[\sum_{t=1}^{T} -\log(p_\theta(s_{t+1}|s_t, \mathbf{z}))\right] + D_{KL}(q_\phi(\mathbf{z}|\tau)||p_\theta(\mathbf{z})). \tag{9.1}$$

We use a prior distribution $p_\theta(\mathbf{z})$ on $\mathbf{z}$ to regularize the learned embeddings; in this study, our prior is the unit Gaussian. By optimizing for the TVAE loss only, we learn an unsupervised version of TREBA. When performing subsequent behavior modeling tasks such as classification, we use the embedding mean, $\mathbf{z}_\mu$.

Task Programming

Task programming is the process by which domain experts create decoder tasks for trajectory self-supervised learning. This process consists of selecting attributes from trajectory data, writing programs, and creating decoder tasks based on the programs (Figure 9.2). Here, domain experts are people with specialized knowledge for studying behavior, such as neuroscientists or sports analysts.

To start, domain experts identify attributes from trajectory data relevant to the behaviors of interest under study. Behavior attributes capture information that is likely relevant to agent behavior, but is not explicitly included in the trajectory states $\{(s_t)\}_{t=1}^{T}$. These attributes represent structured knowledge that domain experts are implicitly or explicitly considering for behavior analysis, such as the distance between two agents, agent velocity, or the relative positioning of agent body parts.

Domain	Behavior Attributes
Mouse	Facing Angle Mouse 1 and 2, Speed Mouse 1 and 2 Nose-Nose Distance, Nose-Tail Distance, Head-Body Angle Mouse 1 and 2 Nose Movement Mouse 1 and 2
Fly	Speed Fly 1 and 2, Fly-Fly Distance Angular Speed Fly 1 and 2, Facing Angle Fly 1 and 2 Min and Max Wing Angles Fly 1 and 2 Major/Minor Axis Ratio Fly 1 and 2

Table 9.1: Behavior Attributes used in Task Programming. We base our programmed tasks in our experiments on these behavior attributes from domain experts in each domain.

Next, domain experts write a program to compute these attributes on trajectory data, which can be done with existing tools such as MARS [36] or SimBA [27]. Algorithm 3 shows a sample program from the mouse social behavior domain, for measuring the "facing angle" between a pair of interacting mice. Each program can be used to construct decoder tasks for self-supervised learning (Section 9.3).

Algorithm 3 Sample Program for Facing Angle

```
Input: centroid of mouse 1 ($x_1$, $y_1$), centroid of mouse 2 ($x_2$, $y_2$), heading of mouse 1 ($\phi_1$)
$x_{\text{diff}} = x_2 - x_1$
$y_{\text{diff}} = y_2 - y_1$
$\theta = \arctan(y_{\text{diff}}, x_{\text{diff}})$
Return $\theta - \phi_1$
```

Our framework is inspired by the data programming paradigm [32], which applies programs to training set creation. In comparison, our framework uses task programming to unify expert-engineered programs, which encode structured expert knowledge, with representation learning.

Working with domain experts in behavioral neuroscience, we created a set of programs to use in studying our approach. The selected programs are a subset of behavior attributes in [36] (for mouse datasets) and a subset of behavior attributes in [13] (for fly datasets). We list the programs used in Table 9.1, and see more details about the programs in the Supplementary Material of [39].

Learning Algorithm

We develop a method to incorporate the programs from domain experts as additional learning signals for TREBA. We consider the following three approaches: (1)

enforcing attribute consistency in generated trajectories (Section 9.3), (2) performing attribute decoding directly (Section 9.3), (3) applying contrastive loss based on program supervision (Section 9.3). Each of these methods applies a different loss on the low-dimensional representation $\mathbf{z}$ of trajectory τ. Any combinations of these decoding tasks can be combined with self-decoding from Section 9.3 to inform the trajectory embedding $\mathbf{z}$.

Attribute Consistency

Let λ be a set of M domain-expert-designed functions measuring agent behavior attributes, such as agent velocity or facing angle. Recall that each $\lambda_j, j = 1...M$ takes as input a trajectory τ, and returns some expert-designed attribute $\lambda_j(\tau)$ computed from that trajectory. For λ_j designed for a single frame, we apply the function to the center frame of τ. Attribute consistency aims to maintain the same behavior attribute labels for the generated trajectory as the original. Let $\tilde{\tau}$ be the trajectory generated by the TVAE given the same initial condition as τ and encoding $\mathbf{z}$.The attribute consistency loss is:

$$\mathcal{L}^{\text{attr}} = \mathbb{E}_{\tau\sim\mathcal{D}}\Bigg[\sum_{j=1}^{M} \mathbb{1}(\lambda_j(\tilde{\tau}) \neq \lambda_j(\tau))\Bigg]. \tag{9.2}$$

Here, we show the loss for categorical λ_j, but in general, λ_j can be continuous and any loss measuring differences between $\lambda_j(\tilde{\tau})$ and $\lambda_j(\tau)$ applies, such as mean squared error. We do not require λ to always be differentiable, and we use the differentiable approximation introduced in [45] to handle non-differentiable λ.

Attribute Decoding

Another option is to decode each attribute $\lambda_j(\tau)$ directly from the learned representation $\mathbf{z}$. Here we apply a shallow decoder f to the learned representation, with decoding loss:

$$\mathcal{L}^{\text{decode}} = \mathbb{E}_{\tau\sim\mathcal{D}}\Bigg[\sum_{j=1}^{M} \mathbb{1}(f(q_\phi(\mathbf{z}_\mu|\tau)) \neq \lambda_j(\tau))\Bigg]. \tag{9.3}$$

Similar to Eq. (9.2), we show the loss for categorical λ_j, however any type of λ may be used.

Contrastive Loss

Lastly, the programmed tasks can be used to supervise contrastive learning of our representation. For a trajectory τ_i, and for each λ_j, positive examples are those trajectories with the same attribute class under λ_j. For λ_j with continuous outputs, we create a discretized $\hat{\lambda}_j$ in which we apply fixed thresholds to divide the output space into classes. For our work, we apply two thresholds for each program such that our classes are approximately equal in size.

We apply a shallow decoder g to the learned representation, and let $\mathbf{g} = g(q_\phi(\mathbf{z}_\mu|\tau))$ represent the decoded representation. We then apply the contrastive loss:

$$\begin{aligned}\mathcal{L}^{\text{cntr.}} = \sum_{i=1}^{B}\sum_{j=1}^{M}\Bigg[\frac{-1}{N_{pos(i,j)}}\sum_{k=1}^{B}\mathbb{1}_{i\neq k}\cdot\mathbb{1}_{\lambda_j(\tau_i)=\lambda_j(\tau_k)} \\ \cdot\log\frac{\exp(\mathbf{g}_i\cdot\mathbf{g}_k/t)}{\sum_{l=1}^{N}\mathbb{1}_{i\neq l}\cdot\exp(\mathbf{g}_i\cdot\mathbf{g}_l/t)}\Bigg],\end{aligned} \tag{9.4}$$

where B is the batch size, $N_{pos(i,j)}$ is the number of positive matches for τ_i with λ_j, and $t > 0$ is a scalar temperature parameter. Our form of contrastive loss supervised by task programming is similar to the contrastive loss in [23] supervised by human annotations. A benefit of task programming is that the supervision from programs can be quickly and scalably applied to unlabelled datasets, as compared to expert supervision which can be time-consuming. We note that the unsupervised version of this contrastive loss is studied in [7], based on previous works such as [28].

Data Augmentation

We can perform data augmentation on trajectory data based on our expert-provided programs. Given the set of all possible augmentations, we define Λ to be the subset of augmentations that are *attribute-preserving*: that is, for all λ_j in the set of programs, $\lambda_j(\tau) = \lambda_j(\Lambda_m(\tau))$ for some augmentation $\Lambda_m \in \Lambda$. An example of a valid augmentation in the mouse domain is reflection of the trajectory data.

All losses presented above can be extended with data augmentation, by replacing τ with $\Lambda_m(\tau)$ in losses. For contrastive loss, adding data augmentation corresponds to extending the batch size to $2B$, with B samples from the original and augmented trajectories.

The augmentations we use in our experiments are reflections, rotations, translations, and a small Gaussian noise on the keypoints (mouse data only). In practice, we add the loss for each decoder with and without data augmentation.

9.4 Experiments

Datasets

We work with datasets from behavioral neuroscience, where there are large-scale, expert-annotated datasets from scientific experiments. We study behavior for the laboratory mouse and the fruit fly, two of the most common model organisms in behavioral neuroscience. For each organism, we first train TREBA using large unannotated datasets: for the mouse domain we use an in-house dataset comprised of approximately 100 hours of recorded diadic social interactions (**Mouse100**), while for the fly domain we use the **Fly vs. Fly** dataset [13] without annotations.

After pre-training TREBA, we evaluate the suitability of our trajectory representation for supervised behavior classification (classifying frame-level behaviors on continuous trajectory data), on three additional datasets:

MARS. The MARS dataset [36] is a recently released mouse social behavior dataset collected in the same conditions as Mouse100. The dataset is annotated by neurobiologists on a frame-by-frame basis for three behaviors: sniff, attack, and mount. We use the provided train, validation, and test split (781k, 352k, and 184k frames, respectively). Trajectories are extracted by the MARS tracker [36].

CRIM13. CRIM13 [4] is a second mouse social behavior dataset manually annotated on a frame-by-frame basis by experts. To extract trajectories, we use a version of the the MARS tracker [36] fine-tuned on pose annotations on CRIM13. We select a subset of videos from which trajectories can be reliably detected for a train, validation and test split of 407k, 96k, and 142k frames, respectively. We evaluated classifier performance on the same three behaviors as MARS (sniff, attack, mount).

CRIM13 is a useful test of the robustness of TREBA trained on Mouse100, as the recording conditions in CRIM13 (image resolution 640×480, frame rate 25Hz, and non-centered cage location) are different from those of Mouse100 (image resolution 1024×570, frame rate 30Hz, and centered cage location).

Fly vs. Fly (Fly). We use the Aggression and Courtship videos from the Fly dataset [13]. These videos record interactions between a pair of flies annotated on a frame-by-frame basis for social behaviors by domain experts. Our train, validation and test split has 1067k, 162k, 322k frames, respectively. We use the trajectories tracked by [13] and evaluate on all behaviors with more than 1000 frames of annotations in the full training set (lunge, wing threat, tussle, wing extension, circle, copulation).

Training and Evaluation Procedure

We use the attribute consistency loss (Section 9.3) and contrastive loss (Section 9.3) to train TREBA using programs. With the same programs, we find that different loss combinations result in similar performance, and that the combination of consistency and contrastive losses performs the best overall. The results for all loss combinations are provided in the Supplementary Material of [39].

For the datasets in the mouse domain (MARS and CRIM13) we train TREBA on Mouse100, with 10 programs provided by mouse behavior domain experts. For the Fly dataset, we train TREBA on the training split of Fly without annotations, with 13 programs provided by fly behavior domain experts. The full list is in Table 9.1. We then use the trained encoder, with pre-trained frozen weights, as a trajectory feature extractor over $T = 21$ frames, where the representation for each frame is computed using ten frames before and after the current frame.

We evaluate our classifiers, with and without TREBA features, using Mean Average Precision (MAP). We compute the mean over behaviors of interest with equal weighting. Our classifiers are shallow fully-connected neural networks on the input features. To determine the relationship between classifier performance and training set size, we sub-sample the training data by randomly sampling trajectories (with lengths of 100 frames) to achieve a desired fraction of the training set size. Sampling was performed to achieve a similar class distribution as the full training set. We train each classifier nine times over three different random selections of the training data for each training fraction (1%, 2%, 5%, 10%, 25%, 50%, 75%, 100%). Additional implementation details are in the Supplementary Material of [39].

Main Results

We evaluate the data efficiency of our representation for supervised behavior classification, by training a classifier to predict behavior labels given both our learned representation and one of either (1) raw keypoints or (2) domain-specific features designed by experts. The TREBA+keypoints evaluation allows us to test the effectiveness of our representation without other hand-designed features, while the TREBA+features evaluation is closer to most potential use cases. The domain-specific features for mice are the trajectory features from [36] and features for flies are the trajectory features from [4]. The input features are a superset of the programs we use in Table 9.1.

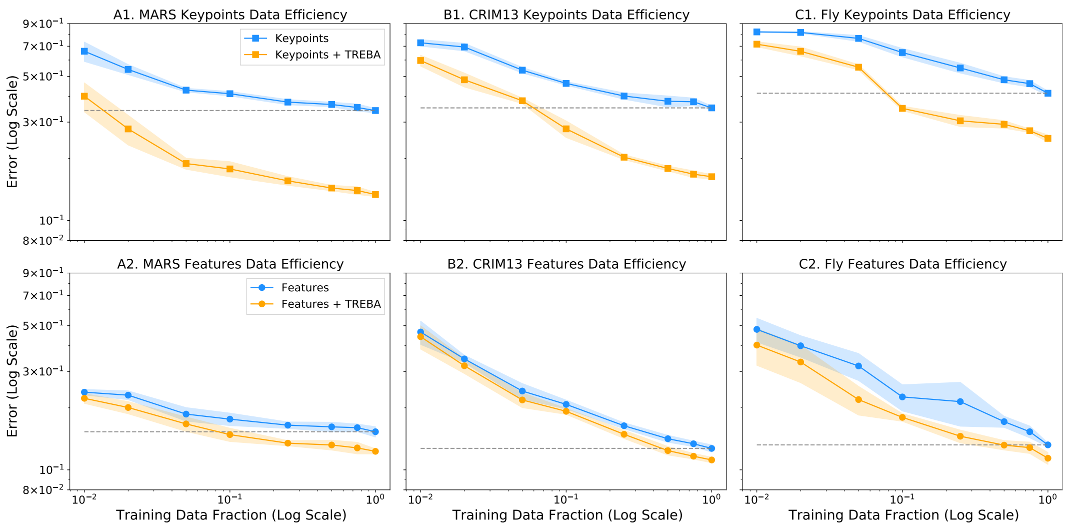


Figure 9.4: Data Efficiency for Supervised Classification. Training data fraction vs. classifier error on MARS (left), CRIM13 (middle) and fly (right). The blue lines represent performance with baseline keypoints and features, and the orange lines are with TREBA. The shaded regions correspond to the classifier standard deviation over nine repeats. The gray dotted line marks the best observed classifier performance when trained on the baseline features (using the full training set). Note the log scale on both the x and y axes.

Our representation is able to improve the data efficiency for both keypoints and domain-specific features, over all evaluated amounts of training data availability (Figure 10.6). We discuss each dataset below:

MARS. Our representation significantly improves classification performance over keypoints alone (Figure 10.6 A1). We achieve the same performance as the full baseline training using only between 1% and 2% of the data. While this result is partially because our representation contains temporal information, we can also observe a significant increase in data efficiency in A2 compared to domain-specific features, which also contains temporal features. Classifiers using TREBA has the same performance as the full baseline training set with around 5% ~ 10% of data (i.e., 10× ~ 20× improved annotation efficiency).

CRIM13. We test the transfer learning ability of our representation on CRIM13, a dataset with different image properties than Mouse100, the training set of TREBA. Our representation achieves the same performance as the baseline training with keypoints using around 5% to 10% of the training data (Figure 10.6 B1). With domain-specific features, TREBA uses 50% of the data annotation to have the same performance as the full training baseline (i.e., 2× improved annotation efficiency). Our representation is able to generalize to a different dataset of the same organism.

Fly. When using keypoints only, our representation requires 10% of the data (Figure 10.6 C1) and for features, our representation requires 50% of the data (Figure 10.6 C2) to achieve the same performance as full baseline training. This corresponds to 2× improved annotation efficiency.

Model Ablations

We perform the following model ablations to better characterize our approach. In this section, percentage error reduction relative to baseline is averaged over all training fractions. Additional results are in the Supplementary Material of [39].

Varying Programmed Tasks. We test the performance of TREBA trained with each single program provided by the domain experts in Table 9.1, and the average, best, and worst performance is visualized in Figure 9.5. On average, representations learned from a single program is better than using features alone, but using all provided programs further improves performance.

For a single program, there could be a large variation in performance depending on the selected program (Figure 9.5). While the best performing single program is close in classifier MAP to using all programs, the worst performing program may increase error, as in MARS and CRIM13. We further tested the performance using more programs.

In the mouse domain, we found that with three randomly selected programs, the variation between runs is much smaller compared to single programs (Supplementary Material of [39]). With three programs, we achieve comparable average error reduction from baseline features to using all programs (MARS: 14.6% error reduction for 3 programs vs. 15.3% for all, CRIM13: 9.2% for 3 programs vs. 9.5% for all). For the fly domain, we found that we needed seven programs to achieve comparable performance (20.7% for 7 programs vs. 21.2% for all).

Varying Decoder Losses. When the programmed tasks are fixed, decoder losses with different combinations of consistency (Section 9.3), decoding (Section 9.3), and contrastive (Section 9.3) loss are similar in performance (Supplementary Material of [39]). Additionally, we evaluate the TREBA framework without programmed tasks, with decoder tasks using trajectory generation and unsupervised contrastive loss. While self-supervised representations are also effective at reducing baseline error, we achieve the best classifier performance using TREBA with programmed

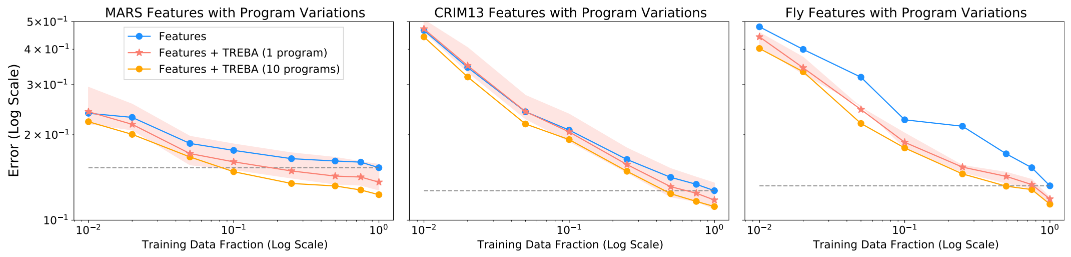


Figure 9.5: Varying Programmed Tasks. Effect of varying number of programmed tasks on classifier data efficiency. The shaded region corresponds to the best and worst classifiers trained using a single programmed task from Table 9.1. The grey dotted line corresponds to the value where the baseline features achieve the best performance (using the full training set).

	Keypoint Error Reduction (%)		
Decoder Loss	MARS	CRIM13	Fly
TVAE	52.2 ± 4.0	34.7 ± 1.5	15.4 ± 2.1
TVAE+ Unsup. Contrast	52.6 ± 3.9	37.4 ± 2.4	20.9 ± 1.7
TVAE+ Contrast+Consist	**55.1 ± 3.0**	**41.1 ± 2.1**	**33.7 ± 1.2**
	Features Error Reduction (%)		
Decoder Loss	MARS	CRIM13	Fly
TVAE	13.7 ± 1.8	8.2 ± 4.6	11.7 ± 4.7
TVAE+ Unsup. Contrast	14.3 ± 2.2	8.9 ± 4.1	16.1 ± 1.7
TVAE+ Contrast+Consist	**15.3 ± 2.1**	**9.5 ± 3.8**	**21.2 ± 4.5**

Table 9.2: Decoder Error Reductions. Percentage error reduction relative to baseline keypoints and domain-specific features for training with different decoder losses for TREBA. The average is taken over all evaluated training fractions.

tasks (Table 9.2). Furthermore, we found that training trajectory representations without self-decoding, using the contrastive loss from [7, 8], resulted in less effective representations for classification (Supplementary Material of [39]).

Data Augmentation. We removed the losses using the data augmentation described in Section 9.3, and found that performance was slightly lower for all datasets than with augmentation. In particular, adding data augmentation decreases error by 1.2% on MARS, 2.5% on CRIM13, and 5.3% on Fly compared to without data augmentation.

Pre-Training Variations The results shown for MARS was obtained with pre-training TREBA on Mouse100, a large in-house mouse dataset with the same image properties as MARS. Figure 9.6 demonstrates the effect of varying TREBA training

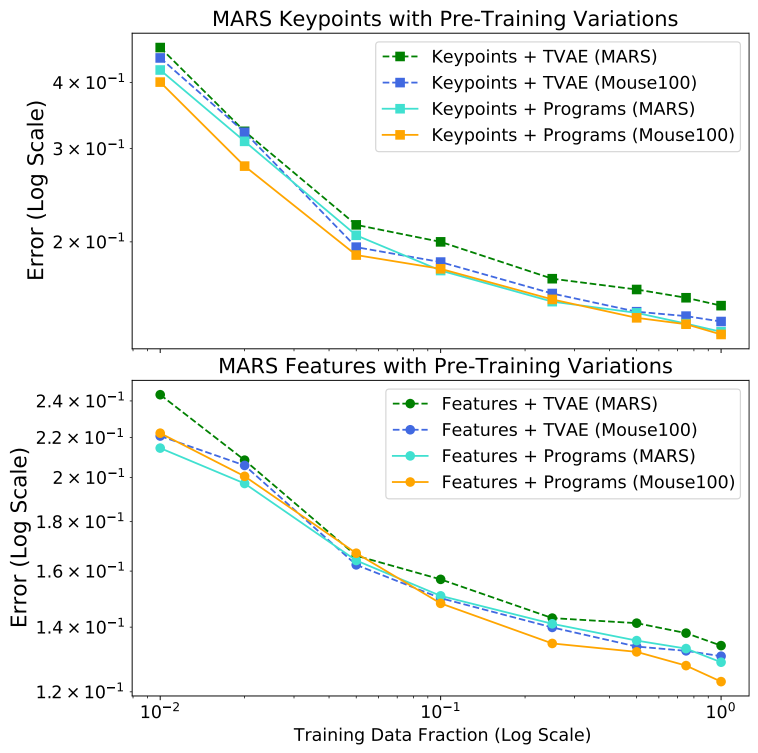


Figure 9.6: Pre-Training Data Variations. Effect of varying pre-training data on classifier data efficiency for the MARS dataset. "TVAE" corresponds to training TREBA with TVAE losses only, and "Programs" corresponds to training with all programs.

data amount with TVAE only and with programs. For both keypoints and features, we observe that TVAE (MARS) has the largest error. We see that error can be decreased by either adding more data (features + TVAE (Mouse100) with 3.9% decrease) or adding task programming (features + Programs (MARS) with 4.4% decrease). Adding both more data and task programming results in an average decrease of 5.7% error relative to TVAE (MARS) and the lowest average error.

Our experiments highlight, and quantify, the tradeoff between task programming and data annotation. The choice of which is more effective will depend on the cost of annotation and the level of expert understanding in identifying behavior attributes. Directions in creating tools to facilitate program creation and data annotation will help further accelerate behavioral studies.

References

[1] David J. Anderson and Pietro Perona. "Toward a Science of Computational Ethology." In: *Neuron* 84.1 (2014), pp. 18–31.

[2] Gordon J. Berman et al. “Mapping the Stereotyped Behaviour of Freely Moving Fruit Flies.” In: *Journal of the Royal Society Interface* 11.99 (2014), p. 20140672.

[3] Brian Broll et al. “Customizing Scripted Bots: Sample-Efficient Imitation Learning for Human-Like Behavior in Minecraft.” In: *AAMAS Workshop on Adaptive and Learning Agents*. 2019.

[4] Xavier P. Burgos-Artizzu et al. “Social Behavior Recognition in Continuous Video.” In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 1322–1329.

[5] Adam J. Calhoun, Jonathan W. Pillow, and Mala Murthy. “Unsupervised Identification of the Internal States that Shape Natural Behavior.” In: *Nature neuroscience* 22.12 (2019), pp. 2040–2049.

[6] Ming-Fang Chang et al. “Argoverse: 3D Tracking and Forecasting with Rich Maps.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8748–8757.

[7] Ting Chen et al. “A Simple Framework for Contrastive Learning of Visual Representations.” In: *International Conference on Machine Learning* (2020).

[8] Ting Chen et al. “Big Self-Supervised Models Are Strong Semi-Supervised Learners.” In: *arXiv preprint arXiv:2006.10029* (2020).

[9] Anthony I. Dell et al. “Automated Image-Based Tracking and Its Application in Ecology.” In: *Trends in Ecology & Evolution* 29.7 (2014), pp. 417–428.

[10] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. “Unsupervised Visual Representation Learning by Context Prediction.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1422–1430.

[11] Carl Doersch and Andrew Zisserman. “Multi-Task Self-Supervised Visual Learning.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2051–2060.

[12] S. E. Roian Egnor and Kristin Branson. “Computational Analysis of Behavior.” In: *Annual Review of Neuroscience* 39 (2016), pp. 217–236.

[13] Eyrun Eyjolfsdottir et al. “Detecting Social Actions of Fruit Flies.” In: *European Conference on Computer Vision*. Springer. 2014, pp. 772–787.

[14] Eyrun Eyjolfsdottir et al. “Learning Recurrent Representations for Hierarchical Behavior Modeling.” In: *International Conference on Learning Representations* (2017).

[15] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. “Unsupervised Representation Learning by Predicting Image Rotations.” In: *ICLR* (2018).

[16] Priya Goyal et al. "Scaling and Benchmarking Self-Supervised Visual Representation Learning." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 6391–6400.

[17] Jacob M. Graving et al. "DeepPoseKit, a Software Toolkit for Fast and Robust Animal Pose Estimation Using Deep Learning." In: *eLife* 8 (2019), e47994.

[18] Tengda Han, Weidi Xie, and Andrew Zisserman. "Video Representation Learning by Dense Predictive Coding." In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2019.

[19] Katja Hofmann. "Minecraft as AI Playground and Laboratory." In: *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*. 2019, pp. 1–1.

[20] Weizhe Hong et al. "Automated Measurement of Mouse Social Behaviors Using Depth Sensing, Video Tracking, and Machine Learning." In: *Proceedings of the National Academy of Sciences* 112.38 (2015), E5351–E5360.

[21] Alexander I. Hsu and Eric A. Yttri. "B-SOiD: An Open Source Unsupervised Algorithm for Discovery of Spontaneous Behaviors." In: *bioRxiv* (2020), p. 770271.

[22] Mayank Kabra et al. "JAABA: Interactive Machine Learning for Automatic Annotation of Animal Behavior." In: *Nature Methods* 10.1 (2013), p. 64.

[23] Prannay Khosla et al. "Supervised Contrastive Learning." In: *arXiv preprint arXiv:2004.11362* (2020).

[24] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. "Revisiting Self-Supervised Visual Representation Learning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1920–1929.

[25] Kevin Luxem et al. "Identifying Behavioral Structure from Deep Variational Embeddings of Animal Motion." In: *bioRxiv* (2020).

[26] Alexander Mathis et al. "DeepLabCut: Markerless Pose Estimation of User-defined Body Parts with Deep Learning." In: *Nature neuroscience* 21.9 (2018), pp. 1281–1289.

[27] Simon R.O. Nilsson et al. "Simple Behavioral Analysis (SimBA)–an Open Source Toolkit for Computer Classification of Complex Social Behaviors in Experimental Animals." In: *BioRxiv* (2020).

[28] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation Learning with Contrastive Predictive Coding." In: *arXiv preprint arXiv:1807.03748* (2018).

[29] AJ Piergiovanni, Anelia Angelova, and Michael S. Ryoo. “Evolving Losses for Unsupervised Video Representation Learning.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 133–142.

[30] Alec Radford et al. *Improving Language Understanding by Generative Pre-Training*. 2018.

[31] Alexander Ratner et al. “Snorkel: Rapid Training Data Creation with Weak Supervision.” In: *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*. Vol. 11. 3. NIH Public Access. 2017, p. 269.

[32] Alexander J. Ratner et al. “Data Programming: Creating Large Training Sets, Quickly.” In: *Advances in Neural Information Processing Systems*. 2016, pp. 3567–3575.

[33] Mirco Ravanelli et al. “Multi-Task Self-Supervised Learning for Robust Speech Recognition.” In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 6989–6993.

[34] John Co-Reyes et al. “Self-Consistent Trajectory Autoencoder: Hierarchical Reinforcement Learning with Trajectory Embeddings.” In: *International Conference on Machine Learning*. 2018.

[35] Aaqib Saeed, Tanir Ozcelebi, and Johan Lukkien. “Multi-Task Self-Supervised Learning for Human Activity Detection.” In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3.2 (2019), pp. 1–30.

[36] Cristina Segalin, Jalani Williams, Tomomi Karigo, May Hui, Moriel Zelikowsky, Jennifer J. Sun, Pietro Perona, David J. Anderson, and Ann Kennedy. “The Mouse Action Recognition System (MARS) Software Pipeline for Automated Analysis of Social Behaviors in Mice.” In: *eLife* 10 (2021), e63720.

[37] Abhinav Shukla, Stavros Petridis, and Maja Pantic. “Does Visual Self-Supervision Improve Learning of Speech Representations?” In: *arXiv preprint arXiv:2005.01400* (2020).

[38] Chen Sun et al. “VideoBERT: A Joint Model for Video and Language Representation Learning.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 7464–7473.

[39] Jennifer J. Sun, Ann Kennedy, Eric Zhan, David J. Anderson, Yisong Yue, and Pietro Perona. “Task Programming: Learning Data Efficient Behavior Representations.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 2876–2885. URL: `https://arxiv.org/pdf/2011.13917.pdf`.

[40] Pei Sun et al. “Scalability in Perception for Autonomous Driving: Waymo Open Dataset.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2446–2454.

[41] Ziyu Wang et al. “Robust Imitation of Diverse Behaviors.” In: *Advances in Neural Information Processing Systems*. 2017, pp. 5320–5329.

[42] Alexander B. Wiltschko et al. “Mapping Sub-Second Structure in Mouse Behavior.” In: *Neuron* 88.6 (2015), pp. 1121–1135.

[43] Raymond A. Yeh et al. “Diverse Generation for Multi-Agent Sports Games.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4610–4619.

[44] Eric Zhan et al. “Generating Multi-Agent Trajectories Using Programmatic Weak Supervision.” In: *International Conference on Learning Representations* (2019).

[45] Eric Zhan et al. “Learning Calibratable Policies Using Programmatic Style-Consistency.” In: *International Conference on Machine Learning* (2020).

[46] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks.” In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017.

Chapter 10

PROGRAM LEARNING

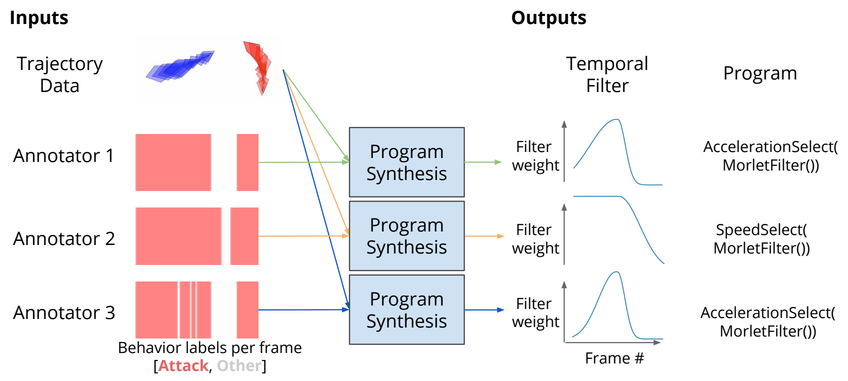


Figure 10.1: Overview of program learning for studying human annotation differences. Given trajectory data and behavior labels, we use program synthesis to learn a programmatic description with temporal filters. These programs can be used to compare differences across annotators.

This chapter is mainly based on the following papers:

[1] Ameesh Shah*, Eric Zhan*, Jennifer J. Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. "Learning Differentiable Programs with Admissible Neural Heuristics." In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 4940–4952. URL: `https://arxiv.org/pdf/2007.12101.pdf`.

[2] Megan Tjandrasuwita, Jennifer J. Sun, Ann Kennedy, Swarat Chaudhuri, and Yisong Yue. "Interpreting Expert Annotation Differences in Animal Behavior." In: *CV4Animals Workshop at the Conference on Computer Vision and Pattern Recognition (CVPR)* (2021). URL: `https://arxiv.org/pdf/2106.06114.pdf`.

Abstract. We study the problem of learning differentiable functions expressed as programs in a domain-specific language. Such programmatic models can offer benefits such as composability and interpretability; however, learning them requires optimizing over a combinatorial space of program "architectures." We frame this optimization problem as a search in a weighted graph whose paths encode top-down

derivations of program syntax. Our key innovation is to view various classes of neural networks as continuous relaxations over the space of programs, which can then be used to complete any partial program. This relaxed program is differentiable and can be trained end-to-end, and the resulting training loss is an approximately admissible heuristic that can guide the combinatorial search. We instantiate our approach on top of the A* algorithm and an iteratively deepened branch-and-bound search, and use these algorithms to learn programmatic classifiers in three sequence classification tasks. Our experiments show that the algorithms outperform state-of-the-art methods for program learning, and that they discover programmatic classifiers that yield natural interpretations and achieve competitive accuracy.

10.1 Introduction

An emerging body of work advocates *program synthesis* as an approach to machine learning. The methods here learn functions represented as programs in symbolic, domain-specific languages (DSLs) [6, 7, 35, 29, 32, 31]. Such symbolic models have a number of appeals: they can be more interpretable than neural models, they use the inductive bias embodied in the DSL to learn reliably, and they use compositional language primitives to transfer knowledge across tasks.

In this chapter, we study how to learn *differentiable* programs, which use structured, symbolic primitives to compose a set of parameterized, differentiable modules. Differentiable programs have recently attracted much interest due to their ability to leverage the complementary advantages of programming language abstractions and differentiable learning. For example, recent work has used such programs to compactly describe modular neural networks that operate over rich, recursive data types [29].

To learn a differentiable program, one needs to induce the program's "architecture" while simultaneously optimizing the parameters of the program's modules. This co-design task is difficult because the space of architectures is combinatorial and explodes rapidly. Prior work has approached this challenge using methods such as greedy enumeration, Monte Carlo sampling, Monte Carlo tree search, and evolutionary algorithms [32, 29, 8]. However, such approaches can often be expensive, due to not fully exploiting the structure of the underlying combinatorial search problem.

We show that the differentiability of programs opens up a new line of attack on this search problem. A standard strategy for combinatorial optimization is to exploit (ideally fairly tight) continuous relaxations of the search space [21, 4, 34, 27, 17, 1, 31]. Optimization in the relaxed space is typically easier and can efficiently guide search algorithms towards good or optimal solutions. In the case of program learning, we propose to use various classes of neural networks as relaxations of partial programs. We frame our problem as searching a graph, in which nodes encode program architectures with missing expressions, and paths encode top-down program derivations. For each partial architecture u encountered during this search, the relaxation amounts to substituting the unknown part of u with a neural network with free parameters. Because programs are differentiable, this network can be trained on the problem's end-to-end loss. If the space of neural networks is an (approximate) proper relaxation of the space of programs (and training identifies a near-optimum neural network), then the training loss for the relaxation can be viewed as an (approximately) admissible heuristic.

We instantiate our approach, called Near (abbreviation for ***N**eural **A**dmissible **R**elaxation*), on top of two informed search algorithms: A* and an iteratively deepened depth-first search that uses a heuristic to direct branching as well as branch-and-bound pruning (Ids-bb). We evaluate the algorithms in the task of learning programmatic classifiers in three behavior classification applications. We show that the algorithms substantially outperform state-of-the-art methods for program learning, and can learn classifier programs that bear natural interpretations and are close to neural models in accuracy.

Contributions. First, we identify a tool — heuristics obtained by training neural relaxations of programs — for accelerating combinatorial searches over differentiable programs. So far as we know, this is the first approach to exploit the differentiability of a programming language in program synthesis. Second, we instantiate this idea using two classic search algorithms. Third, we present promising experimental results in three sequence classification applications.

10.2 Problem Formulation

We view a program in our domain-specific language (DSL) as a pair (α, θ), where α is a discrete *(program) architecture* and θ is a vector of real-valued parameters. The architecture α is generated using a *context-free grammar* [14]. The grammar consists of a set of rules $X \rightarrow \sigma_1 \dots \sigma_k$, where X is a *nonterminal* and $\sigma_1, \dots, \sigma_k$ are either nonterminals or *terminals*. A nonterminal stands for a missing subexpression; a terminal is a symbol that can actually appear in a program's code. The grammar

$$\begin{aligned}\alpha \quad ::= \quad & x \mid c \mid \oplus(\alpha_1, \ldots, \alpha_k) \mid \oplus_\theta(\alpha_1, \ldots, \alpha_k) \mid \textbf{if } \alpha_1 \textbf{ then } \alpha_2 \textbf{ else } \alpha_3 \mid \textbf{sel}_S\, x \\ & \textbf{map } (\textbf{fun } x_1.\alpha_1)\, x \mid \textbf{fold } (\textbf{fun } x_1.\alpha_1)\, c\, x \mid \textbf{mapprefix } (\textbf{fun } x_1.\alpha_1)\, x\end{aligned}$$

Figure 10.2: Grammar of DSL for sequence classification. Here, x, c, $\oplus$, and $\oplus_\theta$ represent inputs, constants, basic algebraic operations, and parameterized library functions, respectively. **fun** $x.e(x)$ represents an anonymous function that evaluates an expression $e(x)$ over the input x. $\textbf{sel}_S$ returns a vector consisting of a subset S of the dimensions of an input x.

starts with an initial nonterminal, then iteratively applies the rules to produce a series of *partial architectures*: sentences made from one or more nonterminals and zero or more terminals. The process continues until there are no nonterminals left, i.e., we have a complete architecture.

The *semantics* of the architecture α is given by a function $[[\alpha]](x, \theta)$, defined by rules that are fixed for the DSL. We require this function to be differentiable in θ. Also, we define a *structural cost* for architectures. Let each rule r in the DSL grammar have a non-negative real cost $s(r)$. The structural cost of α is $s(\alpha) = \sum_{r \in \mathcal{R}(\alpha)} s(r)$, where $\mathcal{R}(\alpha)$ is the multiset of rules used to create α. Intuitively, architectures with lower structural cost are simpler are more human-interpretable.

To define our learning problem, we assume an unknown distribution $D(x, y)$ over inputs x and labels y, and consider the prediction error function $\zeta(\alpha, \theta) = \mathbb{E}_{(x,y)\sim D}[\mathbf{1}([[\alpha]](x, \theta) \neq y)]$, where $\mathbf{1}$ is the indicator function. Our goal is to find an architecturally simple program with low prediction error, i.e., to solve the optimization problem:

$$(\alpha^*, \theta^*) = \arg\min_{(\alpha,\theta)}(s(\alpha) + \zeta(\alpha, \theta)). \tag{10.1}$$

Program Learning for Sequence Classification. Program learning is applicable in many settings; we specifically study it in the sequence classification context. Now we sketch our DSL for this domain. Like many others DSLs for program synthesis [10, 2, 29], our DSL is purely functional. The language has the following characteristics:

- Programs in the DSL operate over two data types: real vectors and sequences of real vectors. We assume a simple type system that makes sure that these types are used consistently.
- Programs use a set of fixed algebraic operations $\oplus$ as well as a "library" of differentiable, parameterized functions $\oplus_\theta$. Because we are motivated by interpretability,

```
map (fun $x_t$.
    if $DistAffine_{[.0217];-.2785}(x_t)$
      then $AccAffine_{[-.0007,.0055,.0051,-.0025];3.7426}(x_t)$
      else $DistAffine_{[-.2143];1.822)}(x_t)$) $x$
```

Figure 10.3: Synthesized program classifying a "sniff" action between two mice in the CRIM13 dataset. *DistAffine* and *AccAffine* are functions that first select the parts of the input that represent distance and acceleration measurements, respectively, and then apply affine transformations to the resulting vectors. In the parameters (subscripts) of these functions, the brackets contain the weight vectors for the affine transformation, and the succeeding values are the biases. The program achieves an accuracy of 0.87 (vs. 0.89 for RNN baseline) and can be interpreted as follows: if the distance between two mice is small, they are doing a "sniff" (large bias in **else** clause). Otherwise, they are doing a "sniff" if the difference between their accelerations is small.

the library used in our current implementation only contains affine transformations. In principle, it could be extended to include other kinds of functions as well.

- Programs use a set of higher-order combinators to recurse over sequences. In particular, we allow the standard **map** and **fold** combinators. To compactly express sequence-to-sequence functions, we also allow a special **mapprefix** combinator. Let g be a function that maps sequences to vectors. For a sequence x, **mapprefix**(g, x) equals the sequence $\langle g(x_{[1:1]}), g(x_{[1:2]}), \ldots, g(x_{[1:n]})\rangle$, where $x_{[1:i]}$ is the i-th prefix of x.

- Programs can use a conditional branching construct. However, to avoid discontinuities, we interpret this construct in terms of a smooth approximation:
$$[[\textbf{if } \alpha_1 > 0 \textbf{ then } \alpha_2 \textbf{ else } \alpha_3]](x, (\theta_1, \theta_2, \theta_3))$$
$$= \sigma(\beta \cdot [[\alpha_1]](x, \theta_1)) \cdot [[\alpha_2]](x, \theta_2) + (1 - \sigma(\beta \cdot [[\alpha_1]](x, \theta_1))) \cdot [[\alpha_3]](x, \theta_3). \tag{10.2}$$
Here, σ is the sigmoid function and β is a temperature hyperparameter. As $\beta \to 0$, this approximation approaches the usual if-then-else construct.

Figure 10.2 summarizes our DSL in the standard Backus-Naur form [33]. Figures 10.3 and 10.4 show two programs synthesized by our learning procedure using our DSL with libraries of domain-specific affine transformations (see the supplementary

```
map (fun $x_t$.
  multiply(add(OffenseAffine($x_t$), BallAffine($x_t$)),
add(OffenseAffine($x_t$), BallAffine($x_t$))) $x$
```

Figure 10.4: Synthesized program classifying the ballhandler for basketball. *OffenseAffine()* and *BallAffine()* are parameterized affine transformations over the XY-coordinates of the offensive players and the ball (see the appendix for full parameters). **multiply** and **add** are computed element-wise. The program structure can be interpreted as computing the Euclidean norm/distance between the offensive players and the ball and suggests that this quantity can be important for determining the ballhandler. On a set of learned parameters (not shown), this program achieves an accuracy of 0.905 (vs. 0.945 for an RNN baseline).

material of [26]). Both programs offer an interpretation in their respective domains, while offering respectable performance against an RNN baseline.

10.3 Program Learning using Near

We formulate our program learning problem as a form of graph search. The search derives program architectures top-down: it begins with the *empty* architecture, generates a series of partial architectures following the DSL grammar, and terminates when a complete architecture is derived.

In more detail, we imagine a graph $\mathcal{G}$ in which:

- The node set consists of all partial and complete architectures permissible in the DSL.
- The *source node* u_0 is the empty architecture. Each complete architecture α is a *goal node*.
- Edges are directed and capture single-step applications of rules of the DSL. Edges can be divided into: (i) *internal edges* (u, u') between partial architectures u and u', and (ii) *goal edges* (u, α) between partial architecture u and complete architecture α. An internal edge (u, u') exists if one can obtain u' by substituting a nonterminal in u following a rule of the DSL. A goal edge (u, α) exists if we can complete u into α by applying a rule of the DSL.
- The cost of an internal edge (u, u') is given by the structural cost $s(r)$, where r is the rule used to construct u' from u. The cost of a goal edge (u, α) is

$s(r) + \zeta(\alpha, \theta^*)$, where $\theta^* = \arg\min_\theta \zeta(\alpha, \theta)$ and r is the rule used to construct α from u.

A path in the graph $\mathcal{G}$ is defined as usual, as a sequence of nodes $u_1, \ldots, u_k$ such that there is an edge (u_i, u_{i+1}) for each $i \in \{1, \ldots, k-1\}$. The cost of a path is the sum of the costs of these edges. Our goal is to discover a least-cost path from the source u_0 to some goal node α^*. Then by construction of our edge costs, α^* is an optimal solution to our learning problem in Eq. Eq. 10.1.

Neural Relaxations as Admissible Heuristics

The main challenge in our search problem is that our goal edges contain rich cost information, but this information is only accessible when a path has been explored until the end. A heuristic function $h(u)$ that can predict the value of choices made at nodes u encountered early in the search can help with this difficulty. If such a heuristic is *admissible* — i.e., underestimates the cost-to-go — it enables the use of informed search strategies such as A* and branch-and-bound while guaranteeing optimal solutions. Our NEAR approach (abbreviation for ***N****eural* ***A****dmissible* ***R****elaxation*) uses neural approximations of spaces of programs to construct a heuristic that is ϵ-close to being admissible.

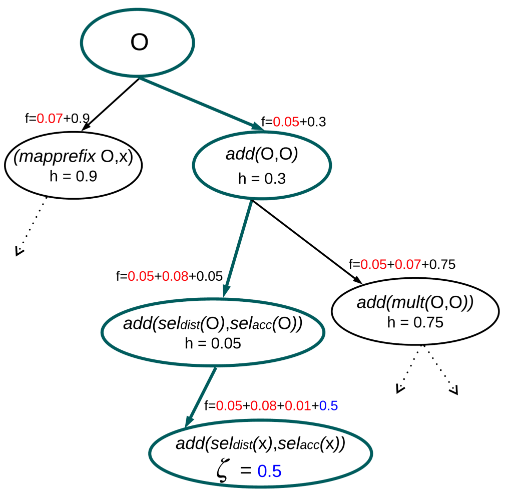


Figure 10.5: An example of program learning formulated as graph search. Structural costs are in red, heuristic values in black, prediction errors ζ in blue, O refers to a nonterminal in a partial architecture, and the path to a goal node returned by A*-NEAR search is in teal.

Let a *completion* of a partial architecture u be a (complete) architecture $u[\alpha_1, \ldots, \alpha_k]$ obtained by replacing the nonterminals in u by suitably typed architectures α_i. Let θ_u be the parameters of u and θ be parameters of the α_i-s. The *cost-to-go* at u is given by:

$$J(u) = \min_{\alpha_1,\ldots,\alpha_k,\theta_u,\theta} ((s(u[\alpha_1, \ldots, \alpha_k] - s(u)) + \zeta(u[\alpha_1, \ldots, \alpha_k], (\theta_u, \theta)) \quad (10.3)$$

where the structural cost $s(u)$ is the sum of the costs of the grammatical rules used to construct u.

To compute a heuristic cost $h(u)$ for a partial architecture u encountered during search, we substitute the nonterminals in u with neural networks parameterized by ω. These networks are *type-correct* — for example, if a nonterminal is supposed to generate subexpressions whose inputs are sequences, then the neural network used in its place is recurrent. We show an example of NEAR used in a program learning-graph search formulation in Figure 10.5.

We view the neurosymbolic programs resulting from this substitution as tuples $(u, (\theta_u, \omega))$. We define a semantics for such programs by extending our DSL's semantics, and lift the function ζ to assign costs $\zeta(u, (\theta_u, \omega))$ to such programs. The heuristic cost for u is now given by:

$$h(u) = \min_{w,\theta} \zeta(u, (\theta_u, \omega)). \tag{10.4}$$

As $\zeta(u, (\theta_u, \omega))$ is differentiable in ω and θ_u, we can compute $h(u)$ using gradient descent.

ϵ-Admissibility. In practice, the neural networks that we use may only form an approximate relaxation of the space of completions and parameters of architectures; also, the training of these networks may not reach global optima. To account for these errors, we consider an approximate notion of admissibility. Many such notions have been considered in the past [13, 21, 28]; here, we follow a definition used by Harris [13]. For a fixed constant $\epsilon > 0$, let an *ϵ-admissible heuristic* be a function $h^*(u)$ over architectures such that $h^*(u) \leq J(u) + \epsilon$ for all u. Now consider any completion $u[\alpha_1, \ldots, \alpha_k]$ of an architecture u. As neural networks with adequate capacity are universal function approximators, there exist parameters ω^* for our neurosymbolic program such that for all $u, \alpha_1, \ldots, \alpha_k, \theta_u$, and θ:

$$\zeta(u, (\theta_u, \omega^*)) \leq \zeta(u[\alpha_1, \ldots, \alpha_k], (\theta_u, \theta)) + \epsilon. \tag{10.5}$$

Because edges in our search graph have non-negative costs, $s(u) \leq s(u[\alpha_1, \ldots, \alpha_k])$, implying:

$$\begin{aligned} h(u) &\leq \min_{\alpha_1,\ldots,\alpha_k,\theta_u,\theta} \zeta(u[\alpha_1, \ldots, \alpha_k], (\theta_u, \theta)) + \epsilon \\ &\leq \min_{\alpha_1,\ldots,\alpha_k,\theta_u,\theta} \zeta(u[\alpha_1, \ldots, \alpha_k], (\theta_u, \theta)) + (s(u[\alpha_1, \ldots, \alpha_k]) - s(u)) + \epsilon = J(u) + \epsilon. \end{aligned} \tag{10.6}$$

In other words, $h(u)$ is ϵ-admissible.

Empirical Considerations. We have formulated our learning problem in terms of the true prediction error $\zeta(\alpha, \theta)$. In practice, we must use statistical estimates of this error. Following standard practice, we use an empirical validation error to choose architectures, and an empirical training error is used to choose module parameters. This means that in practice, the cost of a goal edge (u, α) in our graph is $\zeta^{val}(\alpha, \arg\min_\theta \zeta^{train}(\alpha, \theta))$.

One complication here is that our neural heuristics encode both the completions of an architecture and the parameters of these completions. Training a heuristic on either the training loss or the validation loss will introduce an additional error. Using standard generalization bounds, we can argue that for adequately large training and validation sets, this error is bounded (with probability arbitrarily close to 1) in either case, and that our heuristic is ϵ-admissible with high probability in spite of this error.

Integrating NEAR with Graph Search Algorithms

Algorithm 4 A* Search

```
1: Input: Graph $\mathcal{G}$ with source $u_0$
2: $S := \{u_0\}$; $f(u_0) := \infty$
3: while $S \neq \emptyset$ do
4:     $v := \arg\min_{u \in S} f(u)$
5:     $S := S \setminus \{v\}$
6:     if $v$ is a goal node then
7:         return $v, f_v$
8:     else
9:         for child $u$ of $v$ do
10:            Compute $g(u), h(u), f(u)$
11:            $S := S \cup \{u\}$
```

The NEAR approach can be used in conjunction with any heuristic search algorithm [23] over architectures. Specifically, we have integrated NEAR with two classic graph search algorithms: A^* [21] (Algorithm 4) and an iteratively deepened depth-first search with branch-and-bound pruning (*IDS-BB*) (see Appendix of [26] for more details). Both algorithms maintain a *search frontier* by computing an *f-score* for each node: $f(u) = g(u) + h(u)$, where $g(u)$ is the incurred path cost from the source node u_0 to the current node u, and $h(u)$ is a heuristic estimate of the cost-to-go from node u. Additionally, IDS-BB prunes nodes from the frontier that have a higher f-score than the minimum path cost to a goal node found so far.

ϵ-Optimality. An important property of a search algorithm is *optimality*: when multiple solutions exist, the algorithm finds an optimal solution. Both A* and IDS-

BB are optimal given admissible heuristics. An argument by Harris [13] shows that under heuristics that are ϵ-admissible in our sense, the algorithms return solutions that at most an additive constant ϵ away from the optimal solution. Let C^* denote the optimal path cost in our graph $\mathcal{G}$, and let $h(u)$ be an ϵ-admissible heuristic (Eq. Eq. 10.6). Suppose IDS-BB or A* returns a goal node α_G that does not have the optimal path cost C^*. Then there must exist a node u_O on the frontier that lies along the optimal path and has yet to be expanded. This lets us establish an upper bound on the path cost of α_G:

$$g(\alpha_G) = f(\alpha_G) \leq f(u_O) = g(u_O) + h(u_O) \leq g(u_O) + J(u_O) + \epsilon \leq C^* + \epsilon. \tag{10.7}$$

This line of reasoning can also be extended to the Branch-and-Bound component of the NEAR-guided IDS-BB algorithm. Consider encountering a goal node during search that sets the branch-and-bound upper threshold to be a cost C. In the remainder of search, some node u_p with an f-cost greater than C is pruned, and the optimal path from u_p to a goal node will not be searched. Assuming the heuristic function h is ϵ-admissible, we can set a lower bound on the optimal path cost from u_p, $f(u_p^*)$, to be $C - \epsilon$ by the following:

$$f(u_p^*) = g(u_p) + J(u_p) \geq f(u_p) = g(u_p) + h(u_p) + \epsilon > C = g(u_p) + h(u_p) > C - \epsilon. \tag{10.8}$$

Thus, the IDS-BB algorithm will find goal paths are at worst an additive factor of ϵ more than any pruned goal path.

10.4 Experiments

Datasets for Sequence Classification

For all datasets below, we augment the base DSL in Figure 10.2 with domain-specific library functions that include 1) learned affine transformations over a subset of features, and 2) sliding window feature-averaging functions. Full details, such as structural cost functions used and any pre/post-processing, are provided in the appendix.

CRIM13. The *CRIM13* dataset [3] contains trajectories for a pair of mice engaging in social behaviors, annotated for different actions per frame by behavior experts; we aim to learn programs for classifying actions at each frame for fixed-size trajectories. Each frame is represented by a 19-dimensional feature vector: 4 features capture the xy-positions of the mice, and the remaining 15 features are derived from the positions, such as velocities and distance between mice. We learn programs for two actions that can be identified the tracking features: “sniff” and “other” (“other” is used when there is no behavior of interest occurring). We cut every 100 frames as a trajectory, and in total we have 12404 training, 3077 validation, and 2953 test trajectories.

Fly-vs.-Fly. We use the *Aggression* and *Boy-meets-Boy* datasets within the *Fly-vs.-Fly* environment that tracks a pair of fruit flies and their actions as they interact in different contexts [9]. We aim to learn programs that classify trajectories as one of 7 possible actions displaying aggressive, threatening, and nonthreatening behaviors. The length of trajectories can range from 1 to over 10000 frames, but we segment the data into trajectories with a maximum length of 300 for computational efficiency. The average length of a trajectory in our training set is 42.06 frames. We have 5339 training, 594 validation, and 1048 test trajectories.

Basketball. We use a subset of the basketball dataset from [36] that tracks the movements of professional basketball players. Each trajectory is of length 25 and contains the xy-positions of 5 offensive players, 5 defensive players, and the ball (22 features per frame). We aim to learn programs that can predict which offensive player has the ball (the "ballhandler”) or whether the ball is being passed. In total, we have 18,000 trajectories for training, 2801 for validation, and 2693 for test.

Overview of Baseline Program Learning Strategies

We compare our Near-guided graph search algorithms, A*-Near and Ids-bb-Near, with four baseline program learning strategies: 1) top-down enumeration, 2) Monte-Carlo sampling, 3) Monte-Carlo tree search, and 4) a genetic algorithm. We also compare the performance of these program learning algorithms with an RNN baseline (1-layer LSTM).

Top-down enumeration. We synthesize and evaluate complete programs in order of increasing complexity measured using the structural cost $s(\alpha)$. This strategy is widely employed in program learning contexts [29, 32, 31] and is provably complete. Since our graph $\mathcal{G}$ grows infinitely, our implementation is akin to breadth-first search up to a specified depth.

Monte-Carlo (MC) sampling. Starting from the source node u_0, we sample complete programs by sampling rules (edges) with probabilities proportional to their structural costs $s(r)$. The next node chosen along a path has the best average performance of samples that descended from that node. We repeat the procedure until we reach a goal node and return the best program found among all samples.

	CRIM13-sniff			CRIM13-other			Fly-vs.-Fly			Bball-ballhandler		
	Acc.	F1	Depth	Acc.	F1	Depth	Acc.	F1	Depth	Acc.	F1	Depth
Enum.	.851	.221	3	.707	.762	2	.819	.863	2	.844	.857	6.3
MC	.843	.281	7	.630	.715	1	.833	.852	4	.841	.853	6
MCTS	.745	.338	8.7	.666	.749	1	.817	.857	4.7	.711	.729	8
Genetic	.829	.181	1.7	.727	.768	3	.850	.868	6	.843	.853	6.7
IDS-BB-NEAR	.829	.446	6	.729	.768	1.3	.876	.892	4	.889	.903	8
A*-NEAR	.821	.369	6	.706	.764	2.7	.872	.885	4	.906	.918	8
RNN	.889	.481	-	.756	.785	-	.963	.964	-	.945	.950	-

Table 10.1: Mean accuracy, F1-score, and program depth of learned programs (3 trials). Programs found using our NEAR algorithms consistently achieve better F1-score than baselines and match more closely to the RNN's performance. Our algorithms are also able to search and find programs of much greater depth than the baselines. Experiment hyperparameters are included in the appendix of [26].

Monte-Carlo tree search (MCTS). Starting from the source node u_0, we traverse the graph until we reach a complete program using the UCT selection criteria [16], where the value of a node is inversely proportional to the cost of its children.[1] In the backpropagation step we update the value of all nodes along the path. After some iterations, we choose the next node in the path with the highest value. We repeat the procedure until we reach a goal node and return the best program found.

Genetic algorithm. We follow the formulation in Valkov et al. [29]. In our genetic algorithm, crossover, selection, and mutation operations evolve a population of programs over a number of generations until a predetermined number of programs have been trained. The crossover and mutation operations only occur when the resulting program is guaranteed to be type-safe.

For all baseline algorithms, as well as A*-NEAR and IDS-BB-NEAR, model parameters (θ) were learned with the training set, whereas program architectures (α) were evaluated using the performance on the validation set. Additionally, all baselines (including NEAR algorithms) used F1-score error as the evaluation objective ζ by which programs were chosen. To account for class imbalances, F1-scoring is commonly used as an evaluation metric in behavioral classification domains, such as those considered in our work [9, 3]. Our full implementation is available in [25].

[1]MCTS with this node value definition will visit shallow programs more frequently than MC sampling.

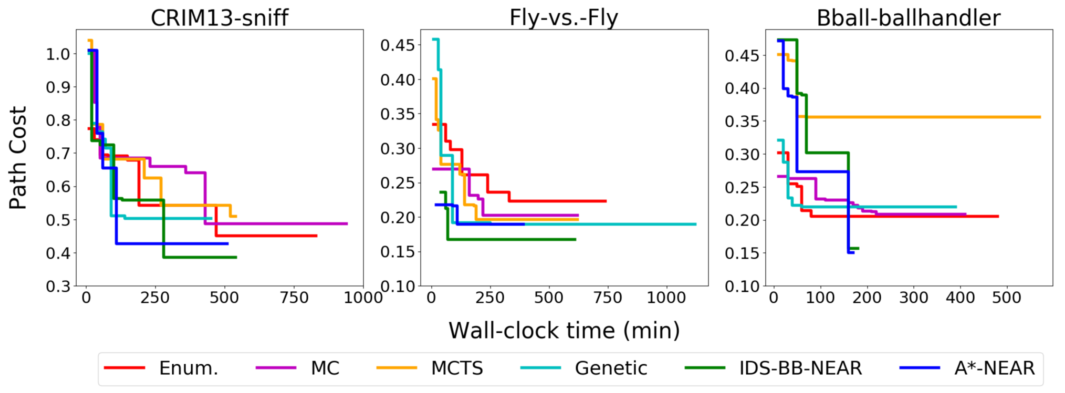
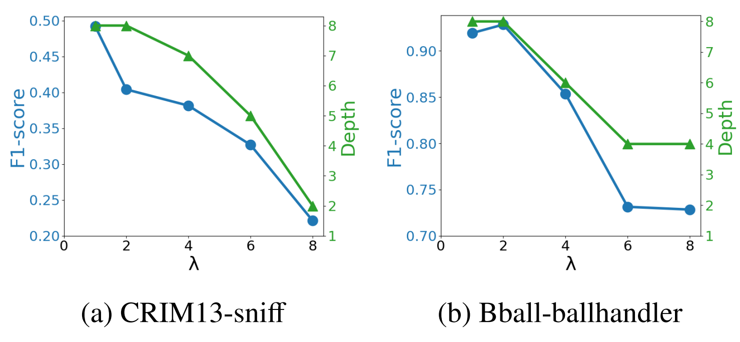


Figure 10.6: Median minimum path cost to a goal node found at a given time, across 3 trials (for trials that terminate first, we extend the plots so the median remains monotonic). A*-Near (blue) and Ids-bb-Near (green) will often find a goal node with a smaller path cost, or find one of similar performance but much faster.



(a) CRIM13-sniff (b) Bball-ballhandler

Figure 10.7: As we increase λ in Eq. Eq. 10.9, we observe that A*-Near will learn programs with decreasing program depth and also decreasing F1-score. This highlights that we can use λ to control the trade-off between structural cost and performance.

Experimental Results

Performance of learned programs. Table 10.1 shows the performance results on the test sets of our program learning algorithms, averaged over 3 seeds. The same structural cost function $s(\alpha)$ is used for all algorithms, but can vary across domains (see Appendix). Our Near-guided search algorithms consistently outperform other baselines in F1-score while accuracy is comparable (note that our ζ does not include accuracy). Furthermore, Near-guided search algorithms are capable are finding deeper and more complex programs that can offer non-trivial interpretations, such as the ones shown in Figures 10.3 and 10.4. Lastly, we verify that our learned programs are comparable with highly expressive RNNs, and see that there is at most a 10% drop in F1-score when using Near-guided search algorithms with our DSL.

Efficiency of Near-guided graph search. Figure 10.6 tracks the progress of each program learning algorithm during search by following the median best path cost

(Eq. 10.1) at a given time across 3 independent trials. For times where only 2 trials are active (i.e., one trial had already terminated), we report the average. Algorithms for each domain were run on the same machine to ensure consistency, and each non-NEAR baseline was set up such to have at least as much time as our NEAR-guided algorithms for their search procedures (see Appendix). We observe that NEAR-guided search algorithms are able to find low-cost solutions more efficiently than existing baselines, while maintaining an overall shorter running time.

Cost-performance trade-off. We can also consider a modification of our objective in Eq. Eq. 10.1 that allows us to use a hyperparameter λ to control the trade-off between structural cost (a proxy for interpretability) and performance:

$$(\alpha^*, \theta^*) = \arg\min_{(\alpha,\theta)}(\lambda \cdot s(\alpha) + \zeta(\alpha, \theta)). \tag{10.9}$$

To visualize this trade-off, we run A*-NEAR with the modified objective Eq. Eq. 10.9 for various values of λ. Note that $\lambda = 1$ is equivalent to our experiments in Table 10.1. Figure 10.7 shows that for the *Basketball* and *CRIM13* datasets, as we increase λ, which puts more weight on the structural cost, the resulting programs found by A*-NEAR search have decreasing F1-scores but are also more shallow. This confirms our expectations that we can control the trade-off between structural cost and performance, which allows users of NEAR-guided search algorithms to adjust to their preferences. Unlike the other two experimental domains, the most performant programs learned in *Fly-vs.-Fly* were relatively shallow, so we omitted this domain as the trade-off showed little change in program depth.

We illustrate the implications of this tradeoff on interpretability using the depth-2 program in Figure 10.8 and the depth-8 program in Figure 10.9, both synthesized for the same task of detecting a “sniff” action in the CRIM13 dataset. The depth-2 program says that a “sniff” occurs if the intruder mouse is close to the right side of the cage and both mice are near the bottom of the cage, and can be seen to apply a *position bias* (regarding the location of the action) on the action. This program is simple, due to the large weight on the structural cost, and has a low F1-score. In contrast, the deeper program in Figure 10.9 has performance comparable to an RNN but is more difficult to interpret. Our interpretation of this program is that it evaluates the likelihood of “sniff” by applying a position bias, then using the velocity of the mice if the mice are close together and not moving fast, and using distance between the mice otherwise.

$$\textbf{mapprefix}\ (\textbf{fun}\ x_t.\textbf{SlidingWindowAverage}(\textit{PositionAffine}(x_t)))\ x$$

Figure 10.8: Synthesized depth 2 program classifying a "sniff" action between two mice in the CRIM13 dataset. The sliding window average is over the last 10 frames. The program achieves F1 score of 0.22 (vs. 0.48 for RNN baseline). This program is synthesized using $\lambda = 8$.

$$\begin{aligned}&\textbf{map}\ (\textbf{fun}\ x_t.\ \textbf{add}(\ \textit{PositionAffine}(x_t),\\ &\qquad \textbf{if}\ (\textbf{add}(\textit{VelocityAffine}(x_t),\ \textit{DistAffine}(x_t)) > 0)\\ &\qquad\ \textbf{then}\ \textit{VelocityAffine}(x_t)\ \textbf{else}\ \textit{DistAffine}(x_t)))\ x\end{aligned}$$

Figure 10.9: Synthesized depth 8 program classifying a "sniff" action between two mice in the CRIM13 dataset. The program achieves F1 score of 0.46 (vs. 0.48 for RNN baseline). This program is synthesized using $\lambda = 1$.

10.5 Interpreting Annotation Differences

Supervised algorithms for animal behavior quantification have become a powerful tool for characterizing the structure of behavior and its regulation by genes and the brain [15, 20]. However, different individuals perceive and describe the world in different ways, and this can create significant inter-annotator and inter-lab differences in the behavioral annotations used to construct such supervised classifiers. Annotator variability has been observed in animal behavior studies, even among experts studying the same behaviors [18, 24]. To improve reproducibility and annotator consensus in behavioral experiments, we propose a method for automatically generating interpretations of human behavior annotations.

Existing behavior classification models are typically black-box models trained to reproduce human annotations. While these models can achieve high accuracy in the hands of individual labs, it is difficult to interpret differences between models or training sets produced by different individuals [24, 20]. Previous studies have proposed methods for post-hoc interpretation of trained models [19, 22], but the large number of dimensions and parameters in modern machine learning models can make it difficult to understand how annotators use specific features to annotate behavior.

To overcome these limitations, we use program synthesis to generate programmatic descriptions from behavior annotations, which can be interpreted without the need for post-hoc analysis. Program synthesis learns symbolic models from domain-specific languages [30, 35, 26, 7]. We introduce a domain-specific language for behavior classification, which includes learnable temporal filters and feature selections to identify behaviorally relevant features of animal movement. We incorporate our setup into an existing program synthesis method [26], described earlier in this Section, to jointly search through the combinatorially large space of program architectures and optimize parameters. Our approach produces a program with temporal filters for modeling expert annotations, which domain experts qualitatively found to be interpretable for behavior analysis.

Approach using Learnable Temporal Filters

We develop a DSL from which program synthesis methods can find interpretable programs, based on the Morlet wavelet [11, 12]. To learn temporal information, our DSL includes a Morlet Filter operation that maps a sequence of vectors to a single vector by taking a weighted sum of the input sequence. The Morlet Filter, denoted by ψ, first does a one-to-one mapping between frames $1, \ldots, n$ in the sliding window to values $x_1, \ldots, x_n$, where $x_i \in [-\pi, \pi]$ $\forall i = 1, \ldots, n$. ψ is then evaluated at each x_i and is defined as:

$$\psi(x; s, w) = e^{-0.5\left(\frac{x}{(s/w)}\right)^2} \cos(wx),$$
$$\text{where } x \in [-\pi, \pi].$$

The Morlet Filter is parameterized by s, w, where w determines the width of the filter and s controls the wavelet frequency. In our experiments, we use a generalization of the symmetric Morlet Filter by allowing the form of the Morlet Filter to differ between the frames preceding and following the predicted frame. Specifically, the left (preceding) Morlet Filter is parameterized by s_1, w_1 whereas the right (following) is parameterized by s_2, w_2, resulting in the asymmetric Morlet Filter that we include in our DSL.

Our DSL also includes affine transformations of the following form, where W is a matrix of weights, x is a feature vector, and b is a learned bias:

$$T(x) = W^T x[i_1, \ldots, i_n] + b.$$

Given a full feature vector x, the transformation selects a subset of features at indices $i_1, \ldots, i_n$ and applies a simple linear layer to the feature subset. For the purpose of interpretability, we limit n to be 1 or 2, i.e., the transformations either select a single feature, or the two same features for the resident and intruder mice.

Within our DSL, the Morlet Filter operation is differentiable with respect to parameters s, w, allowing the shape of the filter to be discovered through gradient optimization. Similarly, the weights and bias W, b of each affine transformation T are amenable to gradient descent.

Disjunctions. Our DSL allows *disjunctions* of two or more Morlet Filter operations. The output of a complete Morlet Filter program (the Morlet Filter applied to a sequence of feature vectors, followed by an affine transformation) is a logit. A disjunction combines the predictions of each filter by summing up the outputted logits. In order to reduce variability in the programs found by the disjunction, we perform separate runs of NEAR to find each filter in the disjunction. Once a filter in the disjunction is found, its weights are frozen when discovering the structure and optimizing the parameters of the subsequent filters. This encourages each subsequent filter to explain variance in the dataset that has not been captured by the previous filters.

Experiments to Compare Annotations

Dataset. We use a subset of the MARS [24] dataset for studying annotator variability, which consists of ten 10-minute videos at 30Hz of socially interacting mice from a standard resident-intruder assay. These videos are independently annotated for three behaviors of interest by each of nine domain experts. As input, we use a subset of domain-specific features from the MARS dataset: 1) for both mice, we compute head-body angle, body axis ratio, speed, acceleration, tangential velocity, social angle, 2) across mice, we compute area ellipse ratio, whether resident is facing intruder, and minimum distance of resident nose to intruder body. Here, we consider two binary classification tasks: interact vs. no-interact, and aggression vs. no-aggression. Interaction is defined as frames on which one mouse is sniffing, attacking, or mounting the other; aggression is defined as periods of high-intensity biting, chasing, or grappling.

Evaluation Procedure

We compare the performance of our discovered programs with the following baselines: 1) Decision Trees, a popular choice for both performance and interpretability; 2) 1D Convolutional Neural Networks, a black-box model that is well-suited for processing temporal signals.

Decision Trees (DT). Decision trees are constructed by finding yes/no questions that split the data into the most homogeneous groups. We implement DTs using XGBoost

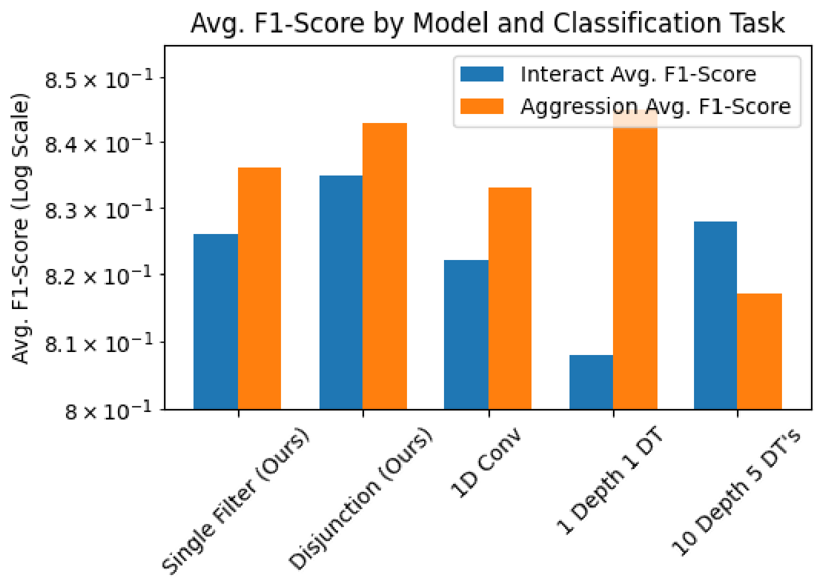


Figure 10.10: **Performance of Models Trained on 100% Training Data.** Bars reflect mean F1 score of each model when trained and tested separately on each of the nine annotators in the MARS dataset.

[5], a popular framework for training tree ensembles, and test tree classifiers of varying complexity. As input to the decision trees, we pass handcrafted temporal features, produced by convolving our 15 behavior features or their first or second derivatives with a Gaussian filter with standard deviations of 8, 30, or 120 frames This produced 135 total features: 15 original features * 3 derivative orders (0, 1, and 2) * 3 filter widths.

1D Convolutional Networks (1D Conv). In a similar manner to a Morlet Filter, a 1D convolutional neural network produces a weighted sum of a given sequence of vectors- however unlike the Morlet Filter, weights are not constrained to have any specific temporal structure. The 1D Conv Net learns a set of weights to convolve with each input feature over time, and the logits from all features are summed for the output predictions.

Evaluation Details. We defined a window of +/- 5 seconds centered about the frame for which behavior was to be predicted, and extracted features of animal poses within this window. We then downsampled data from 30Hz to 6Hz, producing vectors of length 61 for each of the 15 features.

We evaluated all models using the F1-score, defined as the harmonic mean of Precision and Recall. We selected 6 videos for training (106k frames), 2 for val-

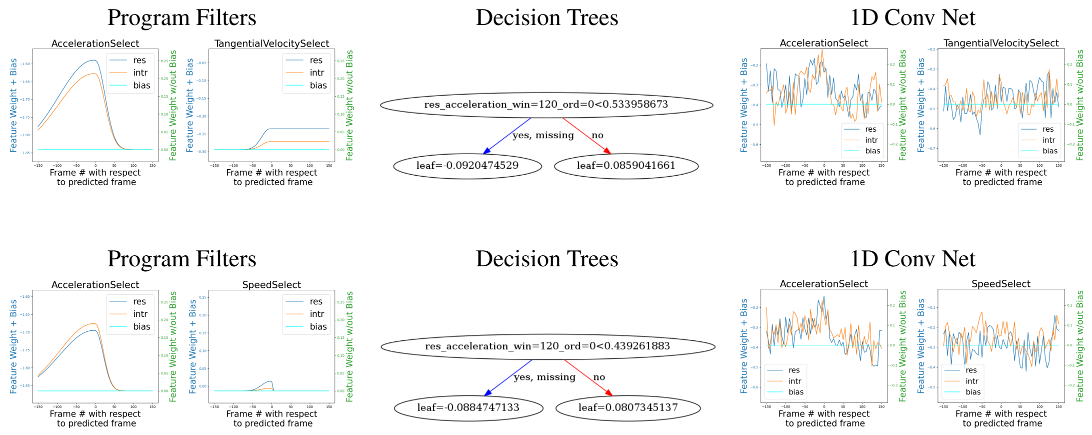


Figure 10.11: Comparing models for two annotators. Each row represents the visualized model trained on aggression vs. nonaggression annotations for one annotator. *Left:* The program filters are from the learned disjunctions, and shows the weight applied at each timestamp for normalized trajectory features from program synthesis. *Center:* Depth 1 decision tree with branches. *Right:* Neural network weights on a subset of input features, matching each annotator's disjunction features.

idation (40k frames), and 2 for test (39k frames). To compare data efficiency, we sub-sampled the training data by randomly sampling trajectories of 1000 frames to achieve desired fractions of the training set size. The sampling also retained a similar class distribution as the full training set. For every data fraction (1%, 10%, 50%), we create three different random samples and train all models three times for each sample. The results are reported on the average across these nine repeats, and across the nine annotators.

Results

Accuracy. Synthesized programs with a disjunction of two filters achieve the highest F1 score for detection of interaction, and are comparable to the Decision Tree (DT) for detection of aggression (Figure 10.10). Programs with a single filter had slightly lower F1 scores compared to the disjunction. For the DTs, the single depth 1 DT is much simpler than 10 depth 5 DTs. Single DT performs better on aggression, which implies that thresholding on one feature is able to classify aggression accurately and the deeper DT is more prone to overfitting. On the other hand, a more complex DT is needed to perform better on interaction.

In terms of data efficiency, disjunctions also remain the highest performing model on interact vs. other (Figure 10.12). On aggression, disjunctions are comparable to the single depth 1 DT. Because of increased model complexity, the 1D Conv

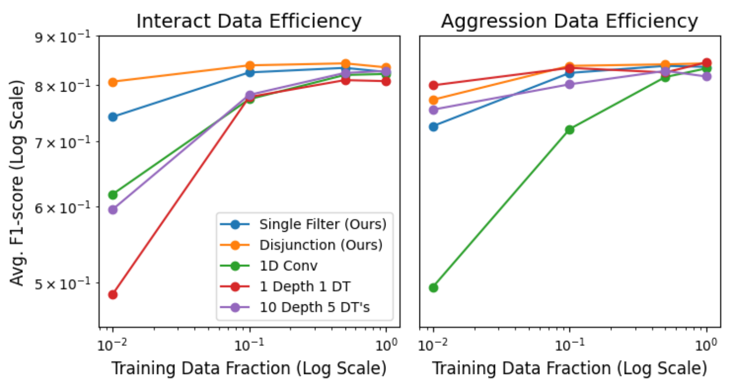


Figure 10.12: Data Efficiency on Behavior Sequence Classification. F1-score averaged across annotators vs. training data fraction on the aggression vs. non-aggression task (left) and interact vs. other task (right).

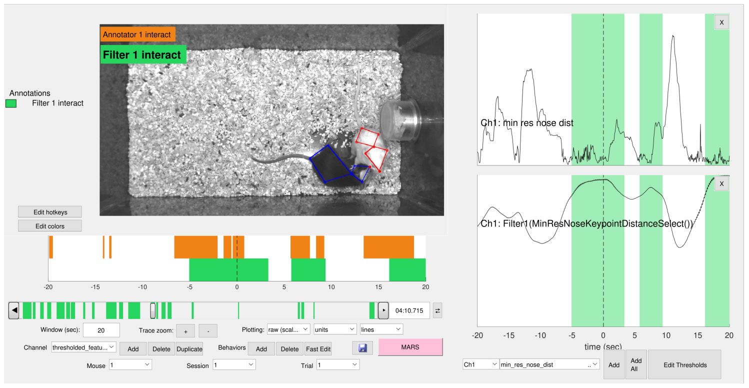


Figure 10.13: Visualization of our learned filters in Bento [24].

Net is generally less data efficient compared to our model. We verified that the variance in performance of both disjunctions and Morlet Filters are either less than or comparable to variance found in the baseline models.

Interpretability. We next visualized our models and baselines (Figure 10.11). All three models include some aspect of temporal filtering of the data, however we argue that visualization and interpretation of this filtering is clearest for the disjunctions. The Conv Net filters appear as noisy versions of the disjunction filters, but without the disjunction filters as reference it is difficult for a domain expert to discern their structure. Filtering in the decision tree is implicit (in the names of the features used), and interpreting the numerical thresholds and leaf values is challenging. In contrast, the smoothness of the disjunction filters makes them easy to read, and their asymmetry around the predicted frame allows them to produce a variety of temporal structures. For domain experts to visualize our model more easily, we also added support for visualization of our trained models and their output within Bento [24] (Figure 10.13).

References

[1] Amitava Bagchi and Ambuj Mahanti. "Admissible Heuristic Search in AND/OR Graphs." In: *Theoretical Computer Science* 24.2 (1983), pp. 207–219.

[2] Matej Balog et al. "DeepCoder: Learning to Write Programs." In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017.

[3] Xavier P. Burgos-Artizzu et al. "Social Behavior Recognition in Continuous Video." In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 1322–1329.

[4] Eugene Charniak and Saadia M. Husain. *A New Admissible Heuristic for Minimal-Cost Proofs*. Brown University, Department of Computer Science, 1991.

[5] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 785–794.

[6] Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. "Sampling for Bayesian Program Learning." In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by Daniel D. Lee et al. 2016, pp. 1289–1297. URL: http://papers.nips.cc/paper/6082-sampling-for-bayesian-program-learning.

[7] Kevin Ellis et al. "Learning to Infer Graphics Programs from Hand-Drawn Images." In: *Advances in Neural Information Processing Systems*. 2018, pp. 6059–6068.

[8] Kevin Ellis et al. "Write, Execute, Assess: Program Synthesis with a REPL." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al. 2019, pp. 9165–9174. URL: http://papers.nips.cc/paper/9116-write-execute-assess-program-synthesis-with-a-repl.

[9] Eyrun Eyjolfsdottir et al. "Detecting Social Actions of Fruit Flies." In: *European Conference on Computer Vision*. Springer. 2014, pp. 772–787.

[10] John K. Feser, Swarat Chaudhuri, and Isil Dillig. "Synthesizing Data Structure Transformations from Input-Output Examples." In: *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*. 2015, pp. 229–239. DOI: 10.1145/2737924.2737977. URL: https://doi.org/10.1145/2737924.2737977.

[11] Dennis Gabor. “Theory of Communication.” In: *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering* 93.26 (1946), pp. 429–441.

[12] Alexandre Grossmann, Richard Kronland-Martinet, and Jean Morlet. “Reading and Understanding Continuous Wavelet Transforms.” In: *Wavelets*. Springer, 1990, pp. 2–20.

[13] Larry R. Harris. “The Heuristic Search Under Conditions of Error.” In: *Artificial Intelligence* 5.3 (1974), pp. 217–234.

[14] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3rd Edition*. Pearson International Edition. Addison-Wesley, 2007. ISBN: 978-0-321-47617-3.

[15] Mayank Kabra et al. “JAABA: Interactive Machine Learning for Automatic Annotation of Animal Behavior.” In: *Nature Methods* 10.1 (2013), p. 64.

[16] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning.” In: *European conference on machine learning*. Springer. 2006, pp. 282–293.

[17] Richard E. Korf. “Recent Progress in the Design and Analysis of Admissible Heuristic Functions.” In: *International Symposium on Abstraction, Reformulation, and Approximation*. Springer. 2000, pp. 45–55.

[18] Xubo Leng et al. “Quantitative Comparison of Drosophila Behavior Annotations by Human Observers and a Machine Learning Algorithm.” In: *bioRxiv* (2020).

[19] Scott M. Lundberg and Su-In Lee. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4765–4774. URL: `http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf`.

[20] Simon R.O. Nilsson et al. “Simple Behavioral Analysis (SimBA)–an Open Source Toolkit for Computer Classification of Complex Social Behaviors in Experimental Animals.” In: *BioRxiv* (2020).

[21] Judea Pearl. “Heuristics: Intelligent Search Strategies for Computer Problem Solving.” In: *Addison Wesley* (1984).

[22] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “"Why Should I Trust You?" Explaining the Predictions of Any Classifier.” In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2016, pp. 1135–1144.

[23] Stuart Russell and Peter Norvig. “Artificial Intelligence: A Modern Approach.” In: (2002).

[24] Cristina Segalin, Jalani Williams, Tomomi Karigo, May Hui, Moriel Zelikowsky, Jennifer J. Sun, Pietro Perona, David J. Anderson, and Ann Kennedy. "The Mouse Action Recognition System (MARS) Software Pipeline for Automated Analysis of Social Behaviors in Mice." In: *eLife* 10 (2021), e63720.

[25] Ameesh Shah, Eric Zhan, and Jennifer Sun. *NEAR Code Repository*. `https://github.com/trishullab/near`. 2020.

[26] Ameesh Shah*, Eric Zhan*, Jennifer J. Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. "Learning Differentiable Programs with Admissible Neural Heuristics." In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 4940–4952. URL: `https://arxiv.org/pdf/2007.12101.pdf`.

[27] David Sontag et al. "Tightening LP Relaxations for MAP Using Message Passing." In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2012.

[28] Richard Anthony Valenzano et al. "Using Alternative Suboptimality Bounds in Heuristic Search." In: *Twenty-Third International Conference on Automated Planning and Scheduling*. 2013.

[29] Lazar Valkov et al. "HOUDINI: Lifelong Learning as Program Synthesis." In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. 2018, pp. 8701–8712. URL: `http://papers.nips.cc/paper/8086-houdini-lifelong-learning-as-program-synthesis`.

[30] Lazar Valkov et al. "Houdini: Lifelong Learning as Program Synthesis." In: *Advances in neural information processing systems*. 2018.

[31] Abhinav Verma et al. "Imitation-Projected Programmatic Reinforcement Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[32] Abhinav Verma et al. "Programmatically Interpretable Reinforcement Learning." In: *International Conference on Machine Learning*. 2018, pp. 5052–5061.

[33] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

[34] Jing Xiang and Seyoung Kim. "A* Lasso for Learning a Sparse Bayesian Network Structure for Continuous Variables." In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by Christopher J. C. Burges et al. 2013, pp. 2418–2426. URL: `http://papers.nips.`

`cc/paper/5174-a-lasso-for-learning-a-sparse-bayesian-network-structure-for-continuous-variables`.

[35] Halley Young, Osbert Bastani, and Mayur Naik. “Learning Neurosymbolic Generative Models via Program Synthesis.” In: *International Conference on Machine Learning (ICML)*. 2019.

[36] Yisong Yue et al. “Learning Fine-Grained Spatial Models for Dynamic Sports Play Prediction.” In: *2014 IEEE International Conference on Data Mining*. IEEE. 2014, pp. 670–679.

Chapter 11

SYNTHESIZING SUPERVISION SOURCES

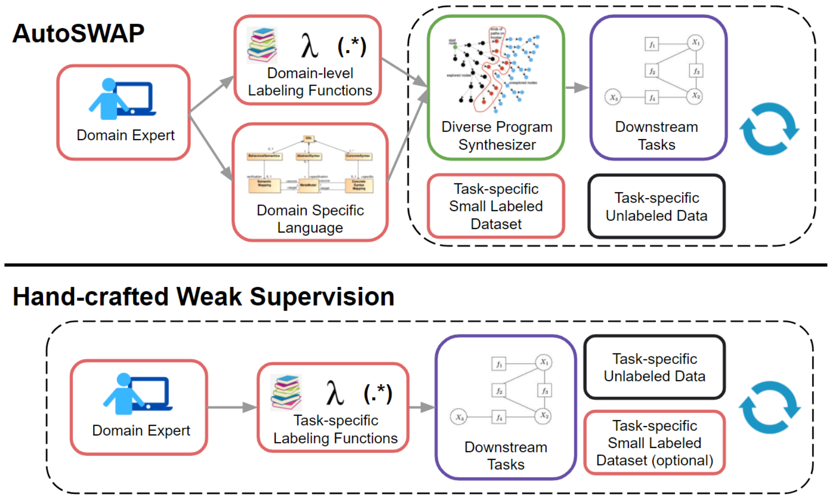

AutoSWAP
Domain Expert
Domain-level Labeling Functions
Domain Specific Language
Diverse Program Synthesizer
Downstream Tasks
Task-specific Small Labeled Dataset
Task-specific Unlabeled Data
Hand-crafted Weak Supervision
Domain Expert
Task-specific Labeling Functions
Downstream Tasks
Task-specific Unlabeled Data
Task-specific Small Labeled Dataset (optional)

Figure 11.1: We present AutoSWAP, a framework for automatically synthesizing diverse sets of *task-level* labeling functions (LFs) with a small labeled dataset and domain knowledge encoded in *domain-level* LFs and a DSL. AutoSWAP significantly reduces labeler effort by automating LF generation.

This chapter is mainly based on the following paper:

[1] Albert Tseng, Jennifer J. Sun, and Yisong Yue. "Automatic Synthesis of Diverse Weak Supervision Sources for Behavior Analysis." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 2211–2220. URL: `https://arxiv.org/pdf/2111.15186.pdf`.

Abstract. Obtaining annotations for large training sets is expensive, especially in settings where domain knowledge is required, such as behavior analysis. Weak supervision has been studied to reduce annotation costs by using weak labels from task-specific labeling functions (LFs) to augment ground truth labels. However, domain experts still need to hand-craft different LFs for different tasks, limiting

scalability. To reduce expert effort, we present AutoSWAP: a framework for automatically synthesizing data-efficient task-level LFs. The key to our approach is to efficiently represent expert knowledge in a reusable domain-specific language and more general domain-level LFs, with which we use state-of-the-art program synthesis techniques and a small labeled dataset to generate task-level LFs. Additionally, we propose a novel structural diversity cost that allows for efficient synthesis of diverse sets of LFs, further improving AutoSWAP's performance. We evaluate AutoSWAP in three behavior analysis domains and demonstrate that AutoSWAP outperforms existing approaches using only a fraction of the data. Our results suggest that AutoSWAP is an effective way to automatically generate LFs that can significantly reduce expert effort for behavior analysis.

11.1 Introduction

In recent years, machine learning has enabled the study of large-scale datasets in many behavior analysis domains, such as neuroscience [22, 26], sports analytics [35, 29], and motion forecasting [7]. However, obtaining labeled data to train models can be difficult and costly, especially when domain expertise is required for annotation, such as for many behavior analysis tasks [22]. One way to reduce annotation cost is through weak supervision, which uses noisy, task-level heuristic "labeling functions" (LFs) to weakly label data. LFs for a specific task (task-level LFs) are supplied by domain experts, and are applied to obtain a set of weak labels. Weakly labeled data can then be used in downstream settings, such as active learning [4] and self-training [15].

While weak supervision has worked well in a wide range of settings [21, 4, 10], it has not been well-explored for behavior analysis tasks. For one, the requirement that LFs must provide *labels* and not, for example, *features* prevents more general domain knowledge from being used [20] (e.g., the behavioral features in [12, 22]). Furthermore, new LFs must be hand-crafted by domain experts for new tasks (such as new behaviors to study), limiting the scalability of manual weak supervision [31]. To address these challenges, we study efficient domain knowledge representations and develop automated weak supervision methods towards reducing annotation bottlenecks in behavior analysis settings.

Our Approach. We propose AutoSWAP (**Auto**matic **S**ynthesized **W**e**A**k Su**P**ervision), a data-efficient framework for automatically generating task-level LFs using a novel diverse program synthesis formulation. As depicted in Figure 11.1, experts provide a domain-specific language (DSL) and domain-level LFs (LFs specific to a domain of tasks) for a given domain, such as mouse behaviors or vehicle motion planning. For each task to be studied in that domain, experts provide a small labeled dataset to specify the task, and AutoSWAP returns a set of structurally diverse task-level LFs that can be used in weakly supervised frameworks. The domain-level LFs (Figure 11.2) provide fine-grained, label-space agnostic "atomic instructions," while the DSL contains abstract structural domain knowledge for composing the more general domain-level LFs into task-level LFs (Figure 11.3). The novel diversity cost enables AutoSWAP to generate structurally diverse LFs, which we and others empirically show outperform structurally homogeneous LFs in downstream tasks [31].

To the best of our knowledge, we are the first to demonstrate the effectiveness of program synthesis for automated LF generation. Existing works for generating LFs include iteratively selecting LFs by repeatedly querying experts for feedback [5] and training exponentially many simple heuristics models [31], which have limitations in scalability and tractability. In contrast, our approach represents domain knowledge in a DSL and domain-level LFs, which can then be used to automatically synthesize LFs for arbitrary tasks in a domain with our diverse program synthesizer.

We evaluate our approach in three behavior analysis domains with both sequential and nonsequential data: mouse [26], fly [12], and basketball player [34] behaviors. In these domains, data collection is expensive and new tasks frequently emerge, highlighting the importance of scalability. The datasets we use are based on agent trajectories, which provide low-dimensional inputs for easily creating domain-level LFs. We show that with existing expert defined domain-level LFs from [22, 12] and a simple DSL, AutoSWAP is capable of synthesizing high quality LFs with very little labeled data. These LFs outperform LFs from existing automatic weak supervision methods [31] and offer a data efficient approach to reducing domain expert effort.

Contributions

- We propose AutoSWAP, which combines program synthesis with weak supervision to scalably and efficiently generate labeling functions.
- We propose a novel program-structural diversity cost that enables AutoSWAP to directly synthesize diverse sets of labeling functions, which we empirically show are more data efficient than purely optimal sets.

- We evaluate AutoSWAP in multiple behavior analysis domains and downstream tasks, and show that AutoSWAP is capable of significantly improving data efficiency and reducing expert cost.

11.2 Related Work

Behavior Analysis. In many domains, such as behavioral neuroscience [22, 17], sports analytics [34, 35], and traffic modeling [9], agent pose and location trajectory data is used for behavior analysis. This data is usually extracted from recorded videos using detectors and pose estimators; for example, we use trajectories from [22], [12], and StatsPerform for our mouse, fly, and basketball datasets, respectively.

To accurately analyze this data for complex behaviors, frame-level behavior labels from domain experts are usually needed. However, annotating large datasets is time-consuming and monotonous [1], motivating methods for label-efficient modeling. For example, self-supervised learning [25] and unsupervised behavior discovery methods [3, 17, 6] aim to learn efficient behavior representations and discover new behaviors, respectively. Our work is complementary to these methods in that this is not a comparison between weak supervision and self-supervision. Rather, we evaluate the merits of our synthesized LFs in the context of weak supervision for learning expert-defined behaviors.

Weak Supervision. Weak supervision with LFs was introduced in the context of data programming [21]. Since then, LFs have been applied in a variety of settings, including for active learning [18, 4] and self-training [15] tasks. Our work is complementary to these works in that we automatically learn LFs that can be used as inputs to existing weakly supervised frameworks. We note that we are not the first to propose learning LFs from a small amount of training data. For example, IWS iteratively proposes rules and queries domain experts in a large-scale feedback loop [5]. More similar to our work, SNUBA [31] trains heuristics models, but does so without domain knowledge and has runtime exponential in the number of features. To the best of our knowledge, we are the first to apply program synthesis to this problem, and our framework outperforms existing model-based methods for learning LFs.

Program Synthesis. Traditionally, programming by example has been used to synthesize programs from a DSL that respect hard constraints on input/output examples [24, 13]. In recent years, a growing number of works have studied synthe-

```
# lambda_1 – whether fly is attacking target
def is_attacking (fly, tgt):
    f2t_angle = atan((tgt.y – fly.y) / (tgt.x – fly.x))
    rel_angle = |fly.abs_angle – f2t_angle|
    return fly.speed > 2 and rel_angle < 0.1

# lambda_2 – ratio of fly wingspan
def wing_ratio(fly, tgt):
    return quantize(fly.wing_x / fly.wing_y, 4)

# lambda_3 – fly speed relative to target speed
def relative_speed (fly, tgt):
    return |fly.speed| / |tgt.speed|
```

Figure 11.2: Domain experts provide domain-level labeling functions, such as the ones above for the fly domain. Some domain-level LFs (λ_1, λ_2) label for specific tasks (and would be considered task-level LFs on their own), while others (λ_3) return features.

sizing programs with soft constraints, such as minimizing a loss function [23, 11, 19, 30]. This relaxed form of program synthesis has been applied to a number of different domains including web information extraction [8], image structure analysis [**ellis2017learning**], and learning interpretable agent policies [32]. Of these works, algorithms that learn differentiable programs, such as [23], have shown great promise in being able to efficiently and simultaneously optimize program architectures and parameters. Here, we use concepts from differentiable program synthesis algorithms to synthesize diverse sets of LFs.

11.3 Methods

We introduce AutoSWAP, a framework for automatically generating diverse sets of task-level LFs. In our framework, domain experts provide a set of domain-level LFs and a DSL of useful relations. For each task to be studied, specified with a small labeled dataset, task-level LFs are automatically generated by the AutoSWAP diverse program synthesizer. These LFs can then be used in downstream applications involving weak supervision. In the following sections, we provide a background of key components in AutoSWAP (Section 11.3), detail the framework (Section 11.3), and describe example downstream applications (Section 11.3).

Background

Domain-level Labeling Functions. In weak supervision, users provide a set of *task-level* hand-crafted heuristics called labeling functions (LFs). LFs can be noisy and abstain from labeling, but LFs must output in downstream task's label space

$\mathcal{Y}$. We relax this requirement in AutoSWAP by allowing domain experts to provide *domain-level* LFs (Figure 11.2). These LFs do not have to output in $\mathcal{Y}$, which reduces LF creation overhead and allows for more expressive LFs. This also allows us to reuse LFs across multiple tasks within the same domain, aiding scalability.

Domain Specific Languages. Domain specific languages (DSLs) define the allowable submodules and structures in synthesized programs, and are a key component of program synthesis algorithms. Many recent works have adopted purely functional DSLs [23], where DSL items are functions that output to the input space of other DSL items or the final output space. In AutoSWAP, domain experts provide a purely functional DSL with program structures that may be useful in generated LFs. We show empirically that even using a very simple DSL in AutoSWAP can result in significant reductions in expert effort.

Differentiable Program Synthesis via Neural Completions and Guided Search. Our program synthesis formulation is based on NEAR, which finds ϵ-optimal differentiable programs using admissible search heuristics [14, 23]. While NEAR is one instantiation of AutoSWAP, our diverse synthesis formulation (Section 11.3) is theoretically compatible with any search-based synthesizer. Here, the DSL $\mathcal{D}$ is a context-free grammar with differentiable variables. Programs are defined by a program architecture α in the context-free language of $\mathcal{D}, \mathrm{CFL}_{\mathcal{D}}$, and a set of real parameters θ, and are denoted by $[[\alpha]](x, \theta) : \mathcal{X} \to \mathcal{Y}$. Synthesizing a program that is optimal w.r.t. a cost function F and dataset $(X, Y) \in (\mathcal{X}, \mathcal{Y})$ is equivalent to

$$(\alpha^*, \theta^*) = \arg\min_{\alpha,\theta} F([[\alpha]](X, \theta), Y). \tag{11.1}$$

To find (α^*, θ^*), we search over $\mathrm{CFL}_{\mathcal{D}}$. This search space is a tree $\mathcal{G}$, where the root node is an empty architecture, interior nodes are incomplete architectures (architectures with unknown components), and leaf nodes are complete architectures. Edges in $\mathcal{G}$ represent single productions from $\mathcal{D}$ between two architectures. We bound the search tree by limiting the search depth to m and "completing" incomplete architectures by substituting unknown components with neural networks ("neural completions").

Since neural completions are differentiable, the minimum cost-to-go (CTG) w.r.t. F of a neural completion can be computed by optimizing the neural completion's parameters. Furthermore, this minimum CTG of a neural completion is an ϵ-admissible heuristic[14] for the true CTG of the corresponding incomplete architecture (proof in [23]). This allows us to use informed search algorithms on $\mathcal{G}$ to

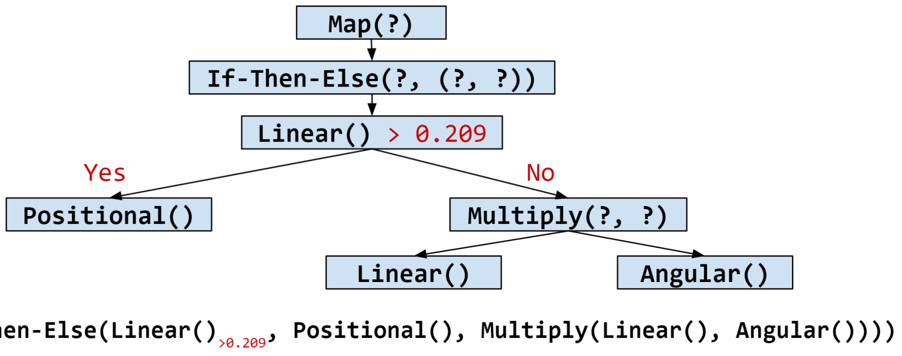


Figure 11.3: A complete program and its tree representation. Each '?' represents one child node function. The depicted program is an actual AutoSWAP LF for the "lunge vs. no behavior" in the Fly domain. The program can be interpreted as *"If the linear speed between the flies is small, classify the angular domain-level LFs of the flies. Otherwise, classify the product of transformations of the linear speed and positional domain-level LFs."* Note the parameters (red) are not included in the structural diversity cost.

find ϵ-optimal solutions to Equation 11.1.

AutoSWAP

Synthesizing Diverse Sets of Programs. Diverse sets of LFs have been shown to improve data efficiency relative to purely optimal sets in downstream applications of weak supervision [31]. This is partly due to diverse sets having improved label coverage (fewer data points where all LFs abstain) [31], and from having more learning signals for the downstream model [27]. The program synthesizer in Section 11.3 can be run repeatedly to obtain a set of purely optimal LFs, but there is no guarantee that the set will be diverse. Here, we introduce a structural diversity cost and admissible heuristic that allows for direct synthesis of diverse sets of programs using informed search algorithms. We empirically show that using the diversity cost improves performance, corroborating [31]'s observations.

Consider a complete program P, which is a composition of variables in $\mathcal{D}$. By construction of $\mathcal{G}$, we can convert P to a tree T_P where each node is a variable in P and a node's children are its input variables (Figure 11.3). Then, given a set of complete programs $\mathcal{P}$ and a complete program P, we define the structural cost $C_{P,\mathcal{P}}$ of P relative to $\mathcal{P}$ as:

$$\frac{1}{C_{P,\mathcal{P}}} = q\left(\frac{1}{\|\mathcal{P}\|}\sum_{P'\in\mathcal{P}} \mathsf{ZSS}(T_P, T_{P'})\right), \tag{11.2}$$

where $q : \mathbb{R} \to \mathbb{R}$ is a user defined monotonically increasing function and ZSS is the

Zhang-Shasha tree edit distance (TED) [36]. Essentially, programs with a higher average TED to the elements of $\mathcal{P}$ incur a lower diversity cost.

Since this structural cost is not defined for incomplete programs or neural completions, $C_{P,\mathcal{P}}$ cannot be used in informed search algorithms. However, the following admissible heuristic $H_{P_I,\mathcal{P}}$ for incomplete programs P_I allows us to create a set of diverse programs by iteratively synthesizing programs and adding them to $\mathcal{P}$.

Lemma 11.3.1. *Let P_I be an incomplete program and T_{P_I} be the tree of its known variables. T_{P_I} is guaranteed to exist by construction of $\mathcal{G}$. Define $H_{P_I,\mathcal{P}}$ as:*

$$U_{P_I,P'} = m - \|P_I\| + \mathtt{ZSS}(T_{P_I}, T_{P'}),$$
$$\frac{1}{H_{P_I,\mathcal{P}}} = q\Big(\frac{1}{\|P\|}\sum_{P'\in\mathcal{P}} U_{P_I,P'}\Big),$$

where $\|P_I\|$ is the number of known variables in P_I. $H_{P_I,\mathcal{P}}$ is an admissible heuristic for the CTG from P_I in $\mathcal{G}$.

Proof. Consider $U_{P_I,P'}$. $m - \|P_I\|$ is an upper bound on the TED between T_{P_I} and the tree of any complete descendant P^* of P_I in $\mathcal{G}$. From the triangle inequality,

$$\begin{aligned} U_{P_I,P'} &= m - \|P_I\| + \mathtt{ZSS}(T_{P_I}, T_{P'}) \\ &\geq \mathtt{ZSS}(T_{P_I}, T_{P^*}) + \mathtt{ZSS}(T_{P_I}, T_{P'}) \\ &\geq \mathtt{ZSS}(T_{P^*}, T_{P'}). \end{aligned}$$

Then, as TEDs are nonnegative, $m \geq \|P_I\|$, and q is nondecreasing, $H_{P_I,\mathcal{P}} \leq C_{P^*,\mathcal{P}}$. Thus, H a admissible heuristic for the structural CTG from P_I. □

AutoSWAP Framework. AutoSWAP uses program synthesis to automate significant parts of the weak supervision pipeline and reduce domain expert effort. Domain experts provide a set of domain-level LFs $\Lambda_m = \{\lambda_i : \mathcal{X} \to \mathcal{Y}_i\}$, a purely functional DSL $\mathcal{D}$, and a small labeled dataset $(X, Y) \in (\mathcal{X}, \mathcal{Y})$ to specify tasks within the domain. In order to use Λ_m when synthesizing programs with $\mathcal{D}$, all λ_i must be added to $\mathcal{D}$. This can be done either by implementing each λ_i with operations from $\mathcal{D}$, or precomputing and selecting $\Lambda_m(X)$ as input features in $\mathcal{D}$; we do the latter in our experiments. With $\mathcal{D}$, AutoSWAP runs the diverse program synthesis algorithm n times to generate a set Λ of n LFs. Λ can then be used in downstream tasks, such as in weak supervision label models to generate weak labels. See Algorithm 5 for a detailed description of AutoSWAP.

Algorithm 5 AutoSWAP.

```
**Input**: $\Lambda_m$, $\mathcal{D}$, labeled dataset $D_L$, # LFs $n$
**Output**: task-level LFs $\Lambda$
$\mathcal{D} \leftarrow$ Combine $\Lambda_M$ and $\mathcal{D}$
$\mathcal{P} \leftarrow \emptyset$
**while** $\|\mathcal{P}\| \leq n$ **do**
    Synthesize $P$ with $\mathcal{D}, D_L, \mathcal{P}$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{P\}$
$\Lambda \leftarrow \mathcal{P}$, return $\Lambda$.
```

Downstream Tasks

We describe two downstream tasks in which weak labels can be used. These examples, which our experiments are based on, are just a subset of the many weakly supervised learning frameworks in existence such as ASTRA [15].

Active Learning. Active learning is a paradigm where the learning algorithm can selectively query for new data to be labeled. Here, we use labels from task-level LFs as additional features for a downstream classifier. The downstream classifier's predictions are used to select data for labeling. To evaluate generated LFs in active learning settings, we consider the performance of downstream classifiers at multiple data amounts. Given a sorted list A of data amounts, at each amount we generate new LFs, train a downstream classifier, and select data points for labeling to form the next batch. An exact description of our active learning setup for AutoSWAP can be found in Algorithm 6.

Algorithm 6 AutoSWAP for Active Learning.

```
**Input**: $\Lambda_m, \mathcal{D}, n$, unlabeled $X_U$, $A$.
Sort $A$ in increasing order.
Randomly select $A_1$ points $X_L$ from $X_U$.
$X_U \leftarrow X_U \setminus X_L$
$Y_L \leftarrow$ Obtain labels for $X_L$.
**for** $i = 1, \ldots, \|A\| - 1$ **do**
    $\Lambda_i \leftarrow$ AutoSWAP($\Lambda_m, \mathcal{D}, (X_L, Y_L), n$).
    $X'_L \leftarrow \begin{bmatrix} X_L & \Lambda_i(X_L) \end{bmatrix}$
    Train downstream classifier $C_i$ with $(X'_L, Y_L)$.
    Select $A_{i+1} - A_i$ points $X'_L$ using max entropy uncertainty
sampling.
    $X_U \leftarrow X_U \setminus X'_L$
    $X_L \leftarrow X_L \cup X'_L$
    $Y_L \leftarrow Y_L \cup$ {Obtain labels for $X'_L$}
```

Weak Supervision. Weak supervision frameworks generally depend on a generative label model weakly label unlabeled samples. Using no ground truth labels, the generative model produces probabilistic estimates ("weak labels") for the true labels Y_U of an unlabeled set X_U by modeling the LF outputs $\Lambda(X_U)$. Weakly labeled data can then be used to augment labeled datasets in downstream tasks.

To evaluate AutoSWAP in weak supervision settings, we start with a small labeled dataset D_L and a list of unlabeled data amounts A. LFs are generated using the small labeled dataset and abstain using the method in [31]. Then, weak labels are generated from these LFs for all unlabeled data using the generative model. For each data amount $A_i \in A$, a random set D_{PL} of A_i weakly labeled data points is selected and the performance of a downstream classifier is measured using the training set $D_L \cup D_{PL}$. An exact description of our weak supervision setup is in Algorithm 7.

Algorithm 7 AutoSWAP for Weak Supervision.

```
**Input**: $\Lambda_m, \mathcal{D}, n$, Labeled $(X_L, Y_L)$, Unlabeled $X_U$, $A$.
$\Lambda \leftarrow$ AutoSWAP($\Lambda_m, \mathcal{D}, (X_L, Y_L), n$).
$\Lambda \leftarrow$ Abstain($\Lambda$) [31]
Sort $A$ in increasing order.
**for** $i = 1, ..., \|A\|$ **do**
    Randomly select $A_i$ points $X_P$ from $X_U$.
    $X'_L \leftarrow X_L \cup X_P$
    $Y'_L \leftarrow Y_L \cup \Lambda(X_P)$
    Train downstream classifier $C_i$ with $(X'_L, Y'_L)$.
```

11.4 Experiments

We evaluate AutoSWAP in multiple real world behavior analysis domains, and show that our framework outperforms existing LF generation methods in weak supervision and active learning settings. Since researchers often study multiple behaviors in a domain [22, 12], we consider each behavior its own task.

Datasets

We use datasets from behavioral neuroscience (mouse and fly behaviors) as well as sports analytics (basketball player trajectories). These datasets include rare behaviors, multi-behavior tasks, and sequential data, making them good representations of real-world behavior analysis tasks. Each dataset contains a train, validation, and test split; the validation split is only used for model checkpoint selection.

Fly vs. Fly (Fly). The fly dataset [12] contains frame-level annotations of videos of interactions between two fruit flies. Our train, validation, and test sets contain 552k, 20k, and 166k frames. We use fly trajectories tracked by FlyTracker [12] and evaluate on 6 behaviors: lunge, wing threat, tussle, wing extension, circle, copulation. This is a multi-label dataset and we report the mean Average Precision (mAP) over binary classification tasks for each behavior. All behaviors except for copulation are rare; lunge, wing threat, and tussle occur in $< 5\%$ of frames, and wing extension and circle occur in $< 1\%$ of frames. The domain-level LFs for this dataset are based on features from [12].

CalMS21 (Mouse). The CalMS21 dataset [26] consists of frame-level pose and behavior annotations from videos of interactions between pairs of mice. We use data from Task 1 (532k train, 20k validation, 119k test) and evaluate on a set of 3 behaviors: attack, investigation, and mount. These behaviors are mutually exclusive and we report the mAP over these classes. We use a subset of the features in [22] as domain-level LFs for this dataset.

Basketball. The Basketball dataset, also used in [34, 23, 35], contains sequences of basketball player trajectories from Stats Perform (18k train, 1k validation, 2.7k test). Labels for which offense player (5 total) had the ball for the majority of the sequence were extracted with [2]. We perform sequential classification in downstream tasks, and report the mAP over each offense player vs. the other 4. Our domain-level LFs include player acceleration, velocity, and position among others. We exclude information about the ball position in the domain-level LFs and data features to focus on analyzing player behaviors.

Baselines

We compare AutoSWAP to two main baselines: student networks from student-teacher training and decision trees from SNUBA [31]. We show that AutoSWAP outperforms both in data efficiency, requiring a fraction of the data to achieve or exceed performance parity. For both baselines, domain-level LFs are incorporated as input features to evaluate the effectiveness of AutoSWAP and not the domain-level LFs themselves. We do not compare against IWS [5], as IWS is a human-in-the-loop LF generation system. We also do not compare against ASTRA [15], as ASTRA is a weak supervision framework for using *task-level* LFs in self training. However, ASTRA can be used as a downstream task for AutoSWAP.

Student Networks Student-teacher training (from knowledge distillation[33]) has been used successfully in self-training. We adopt the concept of student networks by training models with similar capacity as the downstream classifier to serve as LFs. In weak supervision experiments, these student LFs and the label model (Equation 11.3) serve as a teacher model for the downstream classifier.

Decision Trees and SNUBA Decision trees have been shown to be good LFs [31] and offer some degree of interpretability. The SNUBA framework [31] generates a diverse set of decision tree LFs by training $2^k - 1$ decision trees over all feature subsets and then pruning trees based on a diversity and performance metric, where k is the feature dimension of $\mathcal{X}$. Clearly, this is intractable for large k, which is often the case for behavior analysis tasks. Furthermore, SNUBA does not use domain knowledge, instead relying on the complete set of decision trees for data efficiency. In relation to SNUBA, AutoSWAP can be viewed as an scalable alternative to the synthesizer and pruner stages.

Training Setup

Our experimental setup consists of two stages: obtaining LFs, and evaluating generated LFs in downstream tasks. Our downstream tasks include active learning, where LFs are used to select data for labeling, and weak supervision, where LFs generate pseudolabels for unlabeled data points.

Obtaining labeling functions

Synthesized Programs via AutoSWAP. For each domain, we use a simple DSL that includes add, multiply, fold, and differentiable if-then-else (ITE) structures among others. We synthesize programs with our diverse program synthesizer and A* search. Our cost function is the sum of the F_1 cost from [23] and our diversity cost $C_{P,\mathcal{P}}$. We set $q(x)$ to x^2 and m to $\log_2 \|\Lambda_m\|$. Program parameters are trained with weighted cross entropy loss. More information about the exact DSL used is in the Supplementary Materials of [28].

Student Networks. We use neural networks for frame classification tasks and LSTMs for scene classification tasks. To induce diversity in the learned student networks, we take inspiration from [33] and randomly set the size of each layer so the “expected” student network is of similar capacity as the downstream classifier. All student networks are trained using weighted cross entropy loss.

Decision Trees. We fit decision trees using Gini impurity as the split criteria. We limit the depth of decision trees to $\log_2 k$, so the number of nodes is $O(k)$. We select diverse sets of decision trees by pruning a superset of trees based on coverage and performance, similar to how SNUBA does[31]. However, unlike SNUBA, we group our features when generating the superset, as training $2^k - 1$ decision trees is intractable with our datasets.

Downstream Tasks

We use 3 LFs in our main experiments. Experiments with more LFs (5, 7) are in the Supplementary Materials of [28].

Active Learning. As previous described, we evaluate the performance of AutoSWAP at multiple data amounts, selecting additional labeled data with active learning at each amount (Algorithm 6). We use max-entropy uncertainty sampling on downstream classifier outputs to select points for labeling [16]. We use {1000, 2000, 3500, 5000, 7500, 12500, 25000, 50000} frames for the fly and mouse datasets and {500, 1000, 1500, 2000, 3000, 4000, 5000} sequences for the basketball dataset.

Weak Supervision. In our weak supervision experiments, we use factor graph model proposed in [21, 20].

$$p_\theta(Y_U, \Lambda) = Z_\theta^{-1} \exp\Big(\sum_{i=1}^{\|X_U\|} \theta^T \phi_i(\Lambda(X_{U_i}), Y_{U_i})\Big). \tag{11.3}$$

Here, LF accuracies are modeled by factor $\phi_{i,j}^{Acc}(\Lambda, Y_U) = \mathbb{1}\{\Lambda_j(X_{U_i}) = Y_{U_i}\}$, and the proportion of data the LF labels is modeled by $\phi_{i,j}^{Lab}(\Lambda, Y_U) = \mathbb{1}\{\Lambda_j(X_{U_i}) \neq \emptyset\}$.

For the labeled dataset, we use 2000 frames for the fly and mouse datasets, and 500 sequences for the basketball dataset. Our unlabeled data amounts are set to $\{1\times, 2\times, 3\times, 4\times, 5\times\}$ the number of labeled points.

Data Efficiency Results

Active Learning. AutoSWAP LFs are far more data efficient than baseline methods across all datasets, indicating that AutoSWAP is effective in reducing label cost in active learning settings (Figure 11.4). This difference is especially pronounced in the Mouse dataset, where AutoSWAP achieves parity with decision tree LFs with roughly 30× less data. In the Fly dataset, AutoSWAP is consistently $\sim 4\times$ more data efficient than the baselines, and no baseline is able to reach performance parity with AutoSWAP by 50000 samples (9.1% of the entire Fly dataset). We observe a similar trend in the Basketball dataset, with AutoSWAP being $\sim 2\times$ as data efficient. We

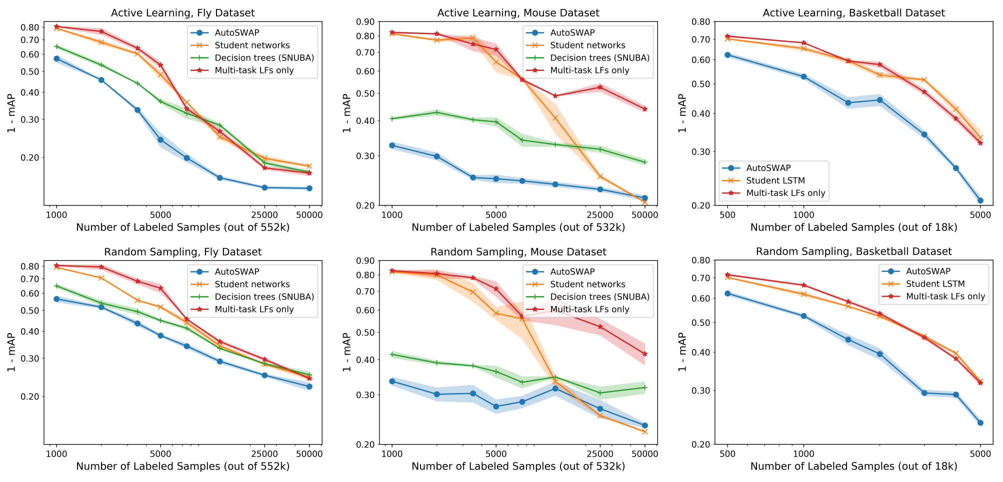


Figure 11.4: AutoSWAP Active Learning Experiments. Each line represents the mean of 5 random seeds for an automatic labeling function method. The shaded region is the standard error of the seeds. As can be seen, AutoSWAP matches or outperforms all baseline methods using only a fraction of the data. Note that all plots are on log-log scales.

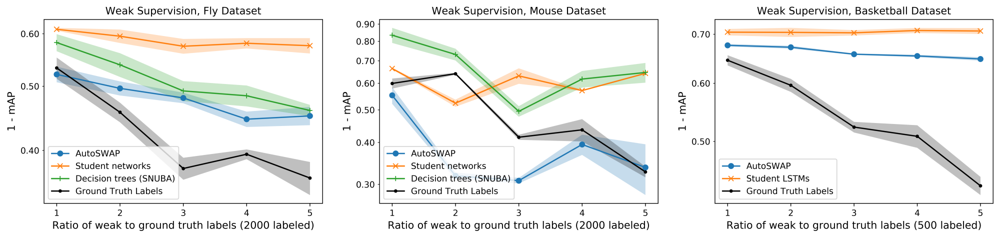


Figure 11.5: AutoSWAP Weak Supervision Experiments. Each line represents the mean of 5 random seeds for an automatic labeling function method. The shaded region is the standard error of the seeds. The gray line shows performance when ground truth labels are used as weak labels. Although it may seem odd that AutoSWAP outperforms ground truth labels in the Mouse dataset, weak labels have been observed to outperform ground truth labels in other works [15]. Note that all plots are on log-log scales.

also observe an improvement in data efficiency even when using random sampling, and note that uncertainty sampling widens the gap between AutoSWAP and the baselines.

While AutoSWAP LFs themselves do not necessarily perform better than baseline LFs when evaluated on their own (see the Supplementary Materials of [28]), they do provide a stronger learning signal for downstream classifiers than the baselines. These data efficiency differences can be attributed in part the structural domain knowledge encoded in the DSL, as the domain-level LFs themselves perform significantly worse. For example, a AutoSWAP LF classifying "lunge vs. no behavior" for the Fly dataset can be seen in Figure 11.3, and the structure of this program cannot be easily approximated with a decision tree or a neural network.

Weak Supervision. Similar to our active learning experiments, we observe that AutoSWAP is more data efficient than the baselines in weak supervision settings (Figure 11.5). We note that the ground truth labels are not a baseline in this setting, as they are essentially an "optimal" case where the weak labels match the ground truth labels.

On the Fly dataset, AutoSWAP generally performs better than both baselines, and on the Mouse and Basketball datasets, no baseline is able to match the performance of AutoSWAP LFs at any evaluated amount of annotated data. AutoSWAP is even able to outperform the ground truth labels in the Mouse dataset at some levels of annotated data, which indicates that the learned LFs are especially informative. Finally, we observe that AutoSWAP generally improves with more weakly labeled data points, which is useful as there is no expert annotation cost to using more weakly labeled data points.

References

[1] David J. Anderson and Pietro Perona. "Toward a Science of Computational Ethology." In: *Neuron* 84.1 (2014), pp. 18–31.

[2] Jenna Wiens Armand McQueen and John Guttag. "Automatically Recognizing On-ball Screens." In: *MIT Sloan Sports Analytics Conference*. 2014.

[3] Gordon J. Berman et al. "Mapping the Stereotyped Behaviour of Freely Moving Fruit Flies." In: *Journal of the Royal Society Interface* 11.99 (2014), p. 20140672.

[4] Samantha Biegel et al. "Active WeaSuL: Improving Weak Supervision with Active Learning." In: *arXiv preprint arXiv:2104.14847* (2021).

[5] Benedikt Boecking et al. "Interactive Weak Supervision: Learning Useful Heuristics for Data Labeling." In: *International Conference on Learning Representations*. 2021. URL: `https://openreview.net/forum?id=IDFQI9OY6K`.

[6] Adam J. Calhoun, Jonathan W. Pillow, and Mala Murthy. "Unsupervised Identification of the Internal States that Shape Natural Behavior." In: *Nature neuroscience* 22.12 (2019), pp. 2040–2049.

[7] Ming-Fang Chang et al. "Argoverse: 3D Tracking and Forecasting with Rich Maps." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8748–8757.

[8] Qiaochu Chen et al. “Web Question Answering with Neurosymbolic Program Synthesis.” In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 2021, pp. 328–343.

[9] James Colyar and John Halkias. “US Highway 101 Dataset.” In: *Federal Highway Administration (FHWA), Tech. Rep. FHWA-HRT-07-030* (2007).

[10] Jared A. Dunnmon et al. “Cross-Modal Data Programming Enables Rapid Medical Machine Learning.” In: *Patterns* 1.2 (2020), p. 100019. ISSN: 2666-3899. DOI: https://doi.org/10.1016/j.patter.2020.100019. URL: https://www.sciencedirect.com/science/article/pii/S2666389920300192.

[11] Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. “Unsupervised Learning by Program Synthesis.” In: (2015).

[12] Eyrun Eyjolfsdottir et al. “Detecting Social Actions of Fruit Flies.” In: *European Conference on Computer Vision*. Springer. 2014, pp. 772–787.

[13] John K. Feser, Swarat Chaudhuri, and Isil Dillig. “Synthesizing Data Structure Transformations from Input-Output Examples.” In: *ACM SIGPLAN Notices* 50.6 (2015), pp. 229–239.

[14] Larry R. Harris. “The Heuristic Search Under Conditions of Error.” In: *Artif. Intell.* 5 (1974), pp. 217–234.

[15] Giannis Karamanolakis et al. “Self-Training with Weak Supervision.” In: *NAACL 2021*. NAACL 2021, May 2021. URL: https://www.microsoft.com/en-us/research/publication/self-training-weak-supervision-astra/.

[16] David Lewis et al. “Heterogeneous Uncertainty Sampling for Supervised Learning.” In: (Dec. 1996).

[17] Kevin Luxem et al. “Identifying Behavioral Structure from Deep Variational Embeddings of Animal Motion.” In: *bioRxiv* (2020).

[18] Mona Nashaat et al. “Hybridization of Active Learning and Data Programming for Labeling Large Industrial Datasets.” In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 46–55.

[19] Emilio Parisotto et al. “Neuro-Symbolic Program Synthesis.” In: *arXiv preprint arXiv:1611.01855* (2016).

[20] Alexander Ratner et al. “Training Complex Models with Multi-Task Weak Supervision.” In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 4763–4771. DOI: 10.1609/aaai.v33i01.33014763. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4403.

[21] Alexander J. Ratner et al. "Data Programming: Creating Large Training Sets, Quickly." In: *Advances in Neural Information Processing Systems*. 2016, pp. 3567–3575.

[22] Cristina Segalin, Jalani Williams, Tomomi Karigo, May Hui, Moriel Zelikowsky, Jennifer J. Sun, Pietro Perona, David J. Anderson, and Ann Kennedy. "The Mouse Action Recognition System (MARS) Software Pipeline for Automated Analysis of Social Behaviors in Mice." In: *eLife* 10 (2021), e63720.

[23] Ameesh Shah*, Eric Zhan*, Jennifer J. Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. "Learning Differentiable Programs with Admissible Neural Heuristics." In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 4940–4952. URL: `https://arxiv.org/pdf/2007.12101.pdf`.

[24] Armando Solar-Lezama et al. "Combinatorial Sketching for Finite Programs." In: *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. 2006, pp. 404–415.

[25] Jennifer J. Sun, Ann Kennedy, Eric Zhan, David J. Anderson, Yisong Yue, and Pietro Perona. "Task Programming: Learning Data Efficient Behavior Representations." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 2876–2885. URL: `https://arxiv.org/pdf/2011.13917.pdf`.

[26] Jennifer J. Sun, Tomomi Karigo, Dipam Chakraborty, Sharada Mohanty, Benjamin Wild, Quan Sun, Chen Chen, David Anderson, Pietro Perona, et al. "The Multi-Agent Behavior Dataset: Mouse Dyadic Social Interactions." In: *Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track* (2021).

[27] Tao Sun and Zhi-Hua Zhou. "Structural Diversity for Decision Tree Ensemble Learning." In: *Frontiers of Computer Science* 12 (Feb. 2018). DOI: `10.1007/s11704-018-7151-8`.

[28] Albert Tseng, Jennifer J. Sun, and Yisong Yue. "Automatic Synthesis of Diverse Weak Supervision Sources for Behavior Analysis." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 2211–2220. URL: `https://arxiv.org/pdf/2111.15186.pdf`.

[29] Karl Tuyls et al. "Game Plan: What AI can do for Football, and What Football can do for AI." In: *Journal of Artificial Intelligence Research* 71 (2021), pp. 41–88.

[30] Lazar Valkov et al. "Houdini: Lifelong Learning as Program Synthesis." In: *Advances in neural information processing systems*. 2018.

[31] Paroma Varma and Christopher Ré. “Snuba: Automating Weak Supervision to Label Training Data.” In: *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*. Vol. 12. 3. NIH Public Access. 2018, p. 223.

[32] Abhinav Verma et al. “Programmatically interpretable reinforcement learning.” In: *International Conference on Machine Learning*. 2018.

[33] Qizhe Xie et al. *Self-training with Noisy Student Improves ImageNet Classification*. 2020. arXiv: `1911.04252 [cs.LG]`.

[34] Yisong Yue et al. “Learning Fine-Grained Spatial Models for Dynamic Sports Play Prediction.” In: *2014 IEEE International Conference on Data Mining*. IEEE. 2014, pp. 670–679.

[35] Eric Zhan et al. “Generating Multi-Agent Trajectories Using Programmatic Weak Supervision.” In: *International Conference on Learning Representations* (2019).

[36] Kaizhong Zhang and Dennis Shasha. “Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems.” In: *SIAM Journal on Computing* 18 (Dec. 1989), pp. 1245–1262. DOI: `10.1137/0218082`.

Chapter 12

NEUROSYMBOLIC PROGRAMMING FOR SCIENCE

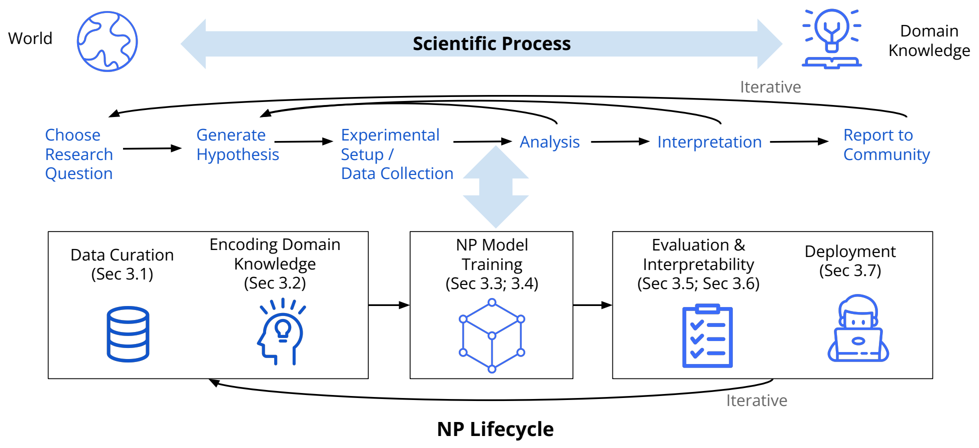


Figure 12.1: Synergy between the scientific and neurosymbolic programming workflow.

This chapter is mainly based on the following paper:

[1] Jennifer J. Sun*, Megan Tjandrasuwita*, Atharva Sehgal*, Armando Solar-Lezama, Swarat Chaudhuri, Yisong Yue, and Omar Costilla-Reyes. "Neurosymbolic Programming for Science." In: *AI for Science Workshop at Neural Information Processing Systems (NeurIPS)* (2022). URL: `https://arxiv.org/pdf/2210.05050.pdf`.

Abstract. Neurosymbolic Programming (NP) techniques have the potential to accelerate scientific discovery. These models combine neural and symbolic components to learn complex patterns and representations from data, using high-level concepts or known constraints. NP techniques can interface with symbolic domain knowledge from scientists, such as prior knowledge and experimental context, to produce interpretable outputs. We identify opportunities and challenges between current NP models and scientific workflows, with real-world examples from behavior analysis in science: to enable the use of NP broadly for workflows across the natural and social sciences.

12.1 Introduction

One of the grand challenges in the artificial intelligence and scientific communities is to find an AI scientist: an artificial agent that can automatically design, test, and infer scientific hypotheses from data. This application poses several distinct challenges for existing learning techniques because of the need to ensure that new theories are consistent with prior scientific knowledge, as well as to enable scientists to reason about the implications of new hypotheses and experimental designs.

The distinct requirements of scientific discovery have pushed the community to explore expressive yet symbolically interpretable techniques such as symbolic regression [4], interpretable machine learning [37, 9, 19, 25], as well as program synthesis [21, 11]. These techniques have helped the community make significant progress in a number of applications, such as those discussed in [16] and [30], but we are still far from solving the grand challenge.

We focus on the opportunities and challenges behind an important class of learning techniques based on *Neurosymbolic Programming* (NP) [3]. These techniques combine neural and symbolic reasoning to build expressive models that incorporate prior expert knowledge and strong constraints on model behavior and structure. NP is capable of producing symbolic representations of theories that can be analyzed and manipulated to answer rich counterfactuals.

NP empowers a new line of attack on the grand AI scientist challenge: represent scientific hypotheses as programs in a *Domain Specific Language* (DSL) and use neurosymbolic program synthesis to automatically discover these programs (Figure 12.1). Users can incorporate complex prior knowledge (e.g., known features and constraints) into the design of the DSL. The NP learning algorithms can then follow classic scientific reasoning principles to find predictive programs. Also, models learned this way are often similar to code that human domain experts write during manual scientific modeling. Collectively, these characteristics enable a transparent and interactive process where an AI system and a human expert collaborate on evidence-based reasoning and the discovery of new scientific facts.

Here, we use behavior analysis as a concrete, illustrative example. We start with an introduction to NP (Section 12.2), then outline challenges and opportunities for future research (Section 12.3).

Behavior analysis as running example. We chose behavior analysis as an example use case for several reasons. Behavioral data is spatiotemporal, which is a common data type across the sciences. Correspondingly, underlying challenges are shared in other domains, from monitoring vital signs to modeling physical systems, to studying the dynamics of chemical reactions. Additionally, behavioral data illustrate common challenges with scientific data. These datasets often contain rare behaviors with noisy and imperfect data and can vary significantly in relevant time scales (e.g., milliseconds vs hours). Datasets also vary across labs, organisms/systems, and experimental setups. Finally, automatic behavior quantification is becoming increasingly crucial in many fields, such as neuroscience, ecology, biology, and healthcare. As computational behavior analysis and neurosymbolic learning are both developing research areas, there are many exciting opportunities to explore at their intersection.

Background on behavior analysis. An important objective of behavior analysis is to quantify behavior from video using continuous or discrete representations. We focus on the case of animal behavior analysis in science [1, 7], where there are diverse organisms and naturalistic behaviors. A common approach is first to perform animal pose tracking from video [24, 28], then categorize behaviors of interest from animal pose [31] (as discussed later in Figure 12.3). From an NP perspective, this approach can be viewed as learning a symbolically interpretable intermediate representation (tracked keypoints).

Existing challenges in behavior analysis. Similar to other scientific fields, data collection and annotation are expensive for behavioral experiments. Analyzing data is also time-consuming and expensive since specialized domain expertise is required for identifying behaviors of interest and extracting knowledge. Models need to interface efficiently with scientists and data at both the inputs and outputs from the scientific process (Figure 12.1). For NP models, leveraging domain expertise in the form of behavioral attributes has been demonstrated to improve data efficiency [34] and interpretability [35].

There is a variety of domain expertise that requires new algorithmic designs to integrate into the NP workflow, such as experimental context, existing ethograms, and scientific spatiotemporal constraints.

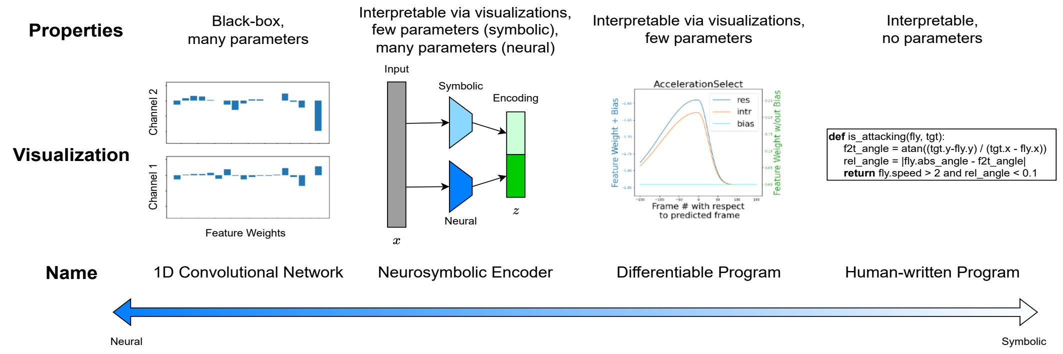
Incorporating such domain knowledge has the potential to enable NP models to be more robust to noisy and imperfect data, and enable new scientific inquiries that were too expensive to study previously. Furthermore, when black-box models are used for studying behavior, it is difficult to diagnose errors and explain model outputs [29]. NP models have the potential to produce symbolic descriptions of behavior (see examples in Chapter 10), which enables experts to connect model interpretations with other parts of the behavior analysis workflow, e.g., describing behavioral differences across different strains of mice. Finally, to enable the use of NP models in real-world science workflows, these models must be scalable and produce robustly reproducible interpretations.

12.2 Neurosymbolic Programming Techniques

Neurosymbolic programs incorporate latent representations from neural networks and symbols that explicitly capture pre-existing human knowledge, and connect these elements using rich architectures that mirror classic models of computation. The programs, assumed to belong to a DSL, are learned using a combination of gradient-based optimization, probabilistic methods, and symbolic techniques.

Anatomy of a Neurosymbolic Program. In general, a neurosymbolic program comprises its discrete architecture and continuous parameters (see examples in Chapter 10). For example, programs can comprise of logical symbolic operations such as "if" statements, as well as functions with continuous parameters. The architecture includes all the discrete symbolic choices that form the structure of the program (such as whether to have an "if" statement and where to place it relative to other operations), and "programming" this architecture is analogous to architecture design in neural networks (e.g., whether to use convolutions, recurrent units, attention, etc.).

Space of Neurosymbolic Programs. The range of NP methods varies in the degree to which they use neural versus symbolic reasoning (Figure 12.2). The two ends of the spectrum correspond to purely neural (a 1D convolutional network) and purely symbolic (a human-written program) models, respectively. The techniques close


Properties
Black-box, many parameters
Interpretable via visualizations, few parameters (symbolic), many parameters (neural)
Interpretable via visualizations, few parameters
Interpretable, no parameters
Visualization
Channel 2
Channel 1
Feature Weights
Input
Symbolic
Encoding
Neural
x
z
AccelerationSelect
res
intr
bias
Feature Weight + Bias
Feature Weight w/out Bias
Frame # with respect to predicted frame
Name
1D Convolutional Network
Neurosymbolic Encoder
Differentiable Program
Human-written Program
Neural
Symbolic

```
def is_attacking(fly, tgt):
    f2t_angle = atan((tgt.y-fly.y) / (tgt.x - fly.x))
    rel_angle = |fly.abs_angle - f2t_angle|
    return fly.speed > 2 and rel_angle < 0.1
```

Figure 12.2: Space of neurosymbolic programming models in behavior analysis, including purely neural (left), purely symbolic (right), and neurosymbolic (two in middle)

to the center are neurosymbolic: the model in the center-left is a neurosymbolic encoder [41], while the model in the center-right is a program with differentiable parameters for behavior analysis [35]. From a definitional perspective, purely neural and purely symbolic programs can be considered special cases of neurosymbolic programs, although we typically do not refer to those as neurosymbolic programs for practical purposes.

To illustrate the strengths and weaknesses of each model in Figure 12.2, assume that we have a scientific hypothesis to test on a dataset. On the right side, the fully symbolic model would involve an expert-written program that encodes the hypothesis in a general programming language. This program requires no learnable parameters, is fully interpretable and, if needed, can be iteratively improved. However, this method is also brittle, and the program must be engineered to handle all the dynamics of the dataset. This is intractable for models with complex dynamics. On the left side, the purely neural model would model the hypothesis directly using the dataset. Such models fit well to the dataset but offer limited interpretability and control over the generated hypothesis, which can make them prone to overfitting and limit generalization. In the middle, neurosymbolic approaches offer a mixture of both neural and symbolic components, and if designed well can inherit both the flexibility of neural networks as well as the structured semantics of symbolic models.

Desiderata for Effective Neurosymbolic Programming. There are two main requirements for effective neurosymbolic programming: having a good DSL, and scalable learning algorithms. The DSL effectively corresponds to the component building blocks from which one can construct a neurosymbolic architecture, and is one of the primary ways that experts inject domain knowledge (i.e., inductive bias) into the learning process. Learning has two aspects: 1) searching for the best neurosymbolic architecture, and 2) optimizing the parameters within a fixed architecture. The former is is analogous to neural architecture search [12], and the latter is analogous to standard parameter optimization in deep learning. We discuss these issues in detail in Section 12.3, and conclude this section with a discussion on differentiable programs that enable differentiable parameter optimization within discrete symbolic architectures.

Differentiable programs. A differentiable program for a programming language is defined as a composition of functions such that the parameters of the function are differentiable. A differentiable program follows the syntax defined by a DSL which can consist of parametric functions (multi-layered perceptrons, linear transformations), algebraic functions (`add`, `multiply`), and programming languages higher-order functions (`map`, `fold`). Furthermore, the composition of these differentiable functions is also differentiable through the chain rule. This property enables resulting programs to be fully differentiable.

NP programs may be difficult to interpret by domain experts [35] focuses on explaining the difference in behavior expert annotations. They replace generic higher-order functions over recursive data structures, e.g., `map` and `fold`, with a differentiable temporal filter operation, the Morlet Filter. The filter models temporal information in a highly data-efficient manner and can be interpreted as a human's impulse response to a given behavioral feature for classification.

12.3 Opportunities and Challenges at the Intersection of Neurosymbolic Learning and Science

We have defined and outlined benefits of the NP framework. However, gaps remain between current NP approaches and practical use cases in science (Figure 12.1). We draw attention to these challenges to encourage the research community to collaborate in the development of new NP methods to increase the synergy with scientific workflows to accelerate scientific discovery.

Dealing with raw, noisy, and imperfect data

Data found in scientific domains provides an opportunity to study NP models with imperfect data in real-world conditions, such as with missing data, experiment noise, and distribution shifts. By incorporating prior knowledge and known constraints in NP models, they have the potential to perform well in the presence of imperfect data. For example, for behavior analysis, neurosymbolic models can automatically learn weak labels from a small amount of annotated examples and apply these trained models to generate weak supervision for a full dataset [36].

These types of imperfect data exist throughout science: missing data in neural recordings due to hardware issues, noise in pose estimators for tracking animal movements, and distribution shifts. An additional source of noise in data is the considerable variability that exists in the labeling generation process, such as annotator subjectivity and ambiguity in category definitions. Furthermore, scientists are often interested in studying rare categories, such as behaviors that may occur in less than 1% of a dataset. NP research [32, 6] leverages the flexibility of neural networks with symbolic domain knowledge; however, there remain challenges in improving model scalability that we have outlined in this section.

Structural discovery. In many scientific workflows, meaningful categories, and structures in raw data may not be clear ahead of time and requires unsupervised or self-supervised learning from data. For example, there are many tools for discovering new behavior categories from data without expert supervision [27]. Zhan* et al. [41] demonstrated that integrating domain knowledge in an NP workflow results in more meaningful discovered categories compared to fully neural methods. In addition to the algorithmic challenges discussed in previous sections, future research work needs to be robust to variations in experimental noise and produce interpretations of discovered structures in the data that are useful in the context of science.

Distribution Shifts. Distribution shifts are common in real-world applications [20]. For typical black-box machine learning models, it is difficult to diagnose and address these errors. NP approaches generally learn interpretable and modular programs, which have the promise to tackle this challenge. For example, in behavior analysis, when the physical behavioral area changes in size, the relative size of mice also changes. This causes errors in behavior classifiers trained in a previously known area, but NP programs can be scaled accordingly to adjust to the new task.

Encoding and Learning Domain Knowledge

The success of NP techniques often depends on how a DSL is defined. However, it is not always clear how to handcraft domain-specific components that work best in a scientific context, and this can be a time-consuming process. *Library learning* proposes algorithms that consolidate common patterns in successful programs and add them iteratively to the current DSL, enabling the program search to discover high-performing programs with little effort.

Library learning for science. In behavior analysis, humans are capable of writing short programs that can improve model learning, such as by designing features and heuristics [31, 36, 13]. However, these programs are greatly limited by their simplicity and may not capture complex behavior. Library learning has the potential to augment human feature design, by synthesizing interpretable programs and inducing high-level DSLs, given low-level, generic primitives. For example, Dreamcoder [10] is a library learning system that has been applied to physics equation discovery. Library learning has also been studied for generative modeling in molecular chemistry [18], which was demonstrated to be able to handle data-limited settings often found in science.

Challenges of library learning for science. In general, it is unclear how library learning can scale to more complex real-world data scientific domains, such as behavior analysis, which often consists of thousands of video frames with noisy data. In addition, it is highly expensive to collect behavior annotations across up to hundreds of behaviors, which is needed to perform traditional library learning. In contrast, current library learning methods have been applied to contexts where each task consists of a few examples, not exceeding hundreds of data points. In addition, the labels are noiseless, as opposed to real-world situations found in behavior analysis [23, 31].

Another challenge is that domain experts still need to interpret solutions generated by the NP library learning system. One promising approach leverages natural language to impose a stronger prior on the program search and the library learning [40], resulting in a more human-interpretable DSL. Additionally, building a smooth interface between expertise in science, program synthesis, neural networks, and probabilistic library learning methods, found in NP, would likely require significant engineering and research efforts (Section 12.3).

Representing informal scientific theories. There is a vast body of knowledge that has been accumulated throughout the span of a given scientific field. Such informal knowledge may not be explicitly represented as a DSL; for instance, behavioral neuroscientists have collected ethograms [14], or natural language descriptions of the functions of species-specific behavior. Other examples include causal relations between phenomena or interventions in an experimental setting. Past work has proposed logical languages capable of representing intuitive theories of causalities [15]. However, capturing all informal and formal knowledge with a single DSL and searching over this space of programs would likely be intractable. Rather, ongoing research in NP focuses on identifying the domain knowledge relevant to a specific subset of scientific problems and distilling such theories into a DSL.

Scalability challenge

From an optimization standpoint, compared to conventional deep learning, the main additional challenge is searching over program architectures. Architecture search is challenging in general and typically leads to combinatorial discrete search space.

Inductive synthesis. A large body of works on program synthesis has focused on *inductive synthesis*, or synthesizing programs from examples [22, 17, 8]. While such a goal is on the surface similar to performing machine learning (ML) with programs as models, a key difference is that ML approaches depend on defining a clear space of models (i.e., neural networks, support vector machines, decision trees) and generalizing to unseen data. In contrast, much work in inductive synthesis considers an arbitrary space of programs and spends significant effort on sample engineering, treating them as noiseless specifications. As a result, inductive synthesis scales poorly with an increase in program length and number of examples.

Scaling NP in science. To tackle scalability in science, models need to handle large and potentially noisy datasets, high-dimensional input space, and a variety of analysis tasks. Recently, NP research [32, 6] propose frameworks that scale to large datasets given an expressive DSL. These works are instantiated in behavior analysis: learning programs on temporal trajectory data to reproduce expert annotations of behavior that contain noisy labels, similar to other scientific data. These works tackle the challenge of discovering programs with parameters, which can be directly optimized through popular gradient optimization techniques. While NP methods provide a means of scaling inductive synthesis to scientific datasets, these techniques often involve combining a discrete search over an exponential space of programs with continuous optimization.

Challenges for enabling scalability. Scaling up program synthesis for neurosymbolic programming is an active field of research. For instance, differentiable program synthesis methods [6] have studied the tradeoff between computation and memory, with heuristics to mitigate memory usage. However, training fully neural models on a GPU is often more efficient than training NP models, which requires searching through an exponentially ample space of symbolic architectures on a CPU. Furthermore, scalability has not been broadly explored for different types of scientific data, such as video recordings, which are much higher dimensional than trajectory data. Finally, the effectiveness of program synthesis may still be limited by the expressivity of a DSL, which requires experts to spend time encoding domain knowledge, such as expert-designed behavior attributes [33] and temporal filters [35] (further discussed in Section 12.3).

Scalability challenges also arise in other work on symbolic regression and interpretable machine learning. For instance, [5] aims to learn exact mathematical relationships between variables by searching a space of mathematical expressions. As another example, Ustun and Rudin [37] aim to learn optimized risk scores within the same modeling language used by clinicians, which leads to an NP-hard optimization problem that they solve using integer programming techniques.

Challenges of optimization of discrete and continuous space in neurosymbolic programs

NP relies on techniques from symbolic program synthesis to facilitate interpretable and verifiable searches over the scientific hypothesis space. However, programs are inherently symbolic, owing to their roots in mathematical logic. This makes modeling phenomena in the continuous domain challenging without modifying the way we interpret programs.

For instance, consider a simple program that is modeled by an if-then-else statement (`if condition do expr1 else do expr2`). The possible behaviors of `condition` are partitioned into two sets — True (`1`) or False (`0`). These sets evaluate to either `expr1` or `expr2`, respectively. However, an NP approach requires reasoning to be differentiable over a *gradient* of possibilities. Discrete programs are inaccurate models for these applications. Specifically, in behavior classification, modeling the "attack" action using a symbolic if-then-else expression would partition the mouse's aggression into a binary set: either always attacking or not attacking at all. What makes more sense is to model "attacking" as a binomial distribution. This requires *relaxing* our symbolic if-then-else to account for a continuous gradient of probabilities from 0% to 100%.

Continuous relaxations. We approach the continuous program optimization problem of the symbolic domain by changing the semantics of the programming language. Specifically, work on *Smooth Interpretation* [2] rewrites discrete functions using their closest smooth mathematical functions. Consecutively, an if-then-else statement would be rewritten as a hyperbolic tangent function with a high temperature. This smoothening is not restricted to a one-dimensional input space and specialized functions. In general, in higher dimensions, we can use Gaussian smoothing to smooth discontinuities. Such relaxations, in conjunction with other program analysis tools, allow gradient descent-based optimizers to converge to optimal programmatic models.

Continuous relaxations enable an approximate interface between neural networks and programming languages, which are essential in the NP framework. For example, in Houdini [38], continuous relaxations enabled the construction of a functional programming language that admits neural networks and higher-order functions. This construction facilitated the high-level transfer of learned concepts across tasks in a lifelong learning setting. In NEAR [32], the interface between neural networks and differentiable programs allowed for measuring the performance of partial programs.

This proved to be an $\epsilon-$admissible heuristic for synthesizing differentiable programs in the behavior analysis setting.

Smooth Interpretation allows positing a differentiable *approximation* for a non-differentiable program. This approximation error introduces a tradeoff between the output precision and optimal trainability of the model. Specifically, under-approximating the non-differentiable components might increase the precision of the differentiable program at the cost of retaining discontinuities in the optimization landscape and converging to a suboptimal model, and vice-versa.

Evaluating Interpretability

The main goal of interpretability is to obtain insights that are understandable and actionable to humans and to assist scientists in their analysis workflow. The following are commonly described properties of explanations found in machine learning [26], that have the potential to improve the interpretability and evaluation of NP workflows: **Compactness or sparsity:** Sparsity generally corresponds to some notion of smallness measurement (a few features or a few parameters); **Completeness:** To measure if the explanation includes all the relevant elements, higher-level concepts needed; **Stability:** To measure the extent that there are explanations similar for similar input; **Actionability:** To allow focusing on only aspects of the model that the user might be able to intervene on; **Modularity:** Explanation can be broken down into understandable parts. To study interpretability of NP models for science, we need datasets and benchmarks to quantify these different dimensions of interpretability across scientific contexts, which is currently an open problem.

Cross-Domain Benchmarking

While many individual fields of science have seen some successes through NP, consolidating underlying generalizable and cross-cutting insights remains another significant open challenge for the scientific and machine learning communities. Towards this, we propose to build initial benchmarks around low-dimensional spatiotemporal data, a setting where NP methods have demonstrated potential [32, 39]. We believe that there are several benefits to gain from developing an NP benchmark for the ML and scientific communities: (1) systematic improvements across broad scientific use cases, (2) comprehensive model evaluations, instead of in domain-specific dimensions, (3) increased awareness of important scientific applications that have not received as much attention from the ML community.

Challenges of benchmarking NP for science. Interpreting programmatic structures requires expert domain knowledge, which can be expensive and time-consuming to obtain. In behavior analysis, evaluating learned programmatic structures requires interactions with experts in the behavioral science community. This imposes a major bottleneck on evaluating outputs. A standardized benchmark will make it easier for the community to convene and interact with a panel of experts. We believe that developing a benchmark for NP pipelines is integral to moving the NP field forward.

The space of NP models is broad. Each algorithm presents a unique methodology for encoding expert knowledge into the NP lifecycle. This requires comparing models on multiple evaluation metrics. However, not all NP algorithms can be systematically evaluated on the same set of metrics. For instance, certain classes of models use stochastic search to discover the programmatic structure and the programs found by such an approach may not be reproducible. Additionally, NP models might exhibit properties that do not have concrete evaluation metrics. For instance, classes of NP algorithms that exhibit robust reproducibility. That is, the model's outputs are reproducible with small perturbations to the input data. However, to the best of our knowledge, defining such a metric quantitatively and objectively remains an open challenge.

The hardware requirements for learning neural representations and symbolic functions are orthogonal. Neural network training is GPU intensive, while program synthesis is CPU intensive. This increases the cost of computation and imposes a barrier to entry for aspiring NP researchers. NP benchmarks need to take the efficiency and performance of training and inference into account.

Cross-Domain Analysis Tools for Scientists

Importance of tools in science. User-friendly tools are important for facilitating the integration of ML models in real-world science workflows but have not been well-explored for NP approaches. For example, numerous tools, based on statistical analysis and ML, have been developed to interface with scientists and facilitate behavior analysis from videos in Pereira, Shaevitz, and Murthy [27]. These tools assist with much of the computational pipeline for behavior classification as outlined in Figure 12.3, and will often provide visual interfaces that visualize relevant raw data such as video, with model outputs, such as pose data and behavior [31]. Enabling similar tools for NP approaches has the potential to benefit existing scientific workflows. For instance, integrating NEAR into a visual interface could provide

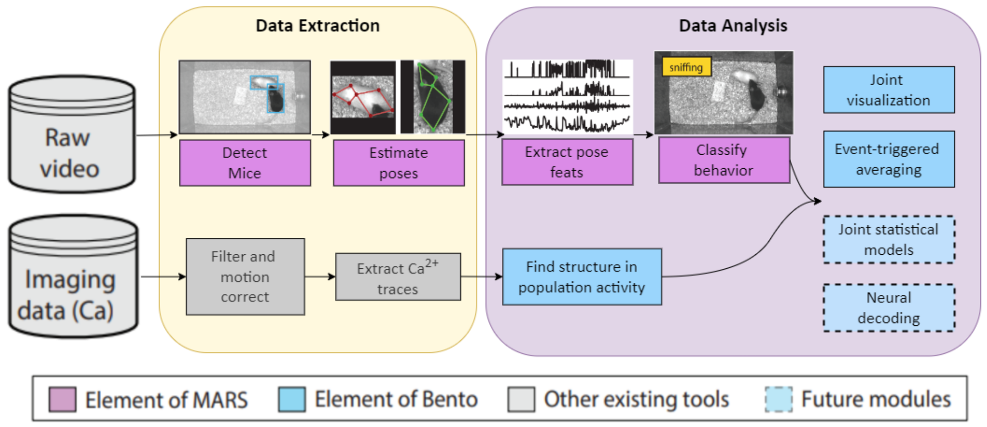


Figure 12.3: Functionalities of MARS and Bento [31] in the behavior analysis pipeline.

scientists with a user-friendly way of generating differentiable programs and means of understanding the programs from NP pipelines. The parameters associated with programmatic primitives are likely to have a much more human interpretation [35] than those found in black-box neural networks.

Challenges of building NP tools. Domain expertise in science varies in structure, from behavioral attributes to visual or textual descriptions, to known dynamics of movement, to knowledge graphs, and generally differs across labs and domains. Furthermore, to measure progress, user evaluations are needed that could offer quantitative or qualitative evidence in NP workflows. Taking the first steps to realize and evaluate the effectiveness of NP algorithms through a human-computer interaction approach may not only improve the scientific pipeline but also yield new algorithmic directions on combining NP with more traditional human-in-the-loop methods, such as active learning.

12.4 Discussion

Neurosymbolic programming offers the promise to accelerate scientific discovery and optimize scientific discovery end-to-end. The benefits are in its ability to incorporate prior knowledge and the symbolic nature of the solutions, essential scientific workflows. However, challenges still remain in scalability and optimization stability of these approaches, comprehensive evaluations, and deployment in the form of tools. In this Chapter, we have demonstrated the opportunities and challenges of neurosymbolic programming in a concrete scientific application, behavior analysis. A key promise of neurosymbolic programming is to provide a set of unifying principles in interpretable machine learning and prior scientific literature. We invite the science and computer science communities to adopt these methods in their scientific workflow and to contribute to the research to advance NP techniques for science due to the unique benefit to these communities.

References

[1] David J. Anderson and Pietro Perona. "Toward a Science of Computational Ethology." In: *Neuron* 84.1 (2014), pp. 18–31.

[2] Swarat Chaudhuri and Armando Solar-Lezama. "Smooth Interpretation." In: *ACM Sigplan Notices* 45.6 (2010), pp. 279–291.

[3] Swarat Chaudhuri et al. "Neurosymbolic Programming." In: *Foundations and Trends® in Programming Languages* 7.3 (2021), pp. 158–243.

[4] Miles Cranmer. *PySR: Fast & Parallelized Symbolic Regression in Python/Julia*. 2020.

[5] Miles Cranmer et al. "Discovering Symbolic Models from Deep Learning with Inductive Biases." In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 17429–17442. (Visited on 09/27/2022).

[6] Guofeng Cui and He Zhu. "Differentiable Synthesis of Program Architectures." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 11123–11135.

[7] Sandeep Robert Datta et al. "Computational Neuroethology: A Call to Action." In: *Neuron* 104.1 (2019), pp. 11–24.

[8] Jacob Devlin et al. "RobustFill: Neural Program Learning under Noisy I/O." en. In: *Proceedings of the 34th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2017, pp. 990–998. (Visited on 09/26/2022).

[9] Finale Doshi-Velez and Been Kim. "Towards a Rigorous Science of Interpretable Machine Learning." In: *arXiv preprint arXiv:1702.08608* (2017).

[10] Kevin Ellis et al. "DreamCoder: Growing Generalizable, Interpretable Knowledge With Wake-Sleep Bayesian Program Learning." In: *arXiv preprint arXiv:2006.08381* (2020).

[11] Kevin Ellis et al. "Synthesizing Theories of Human Language with Bayesian Program Induction." In: *Nature Communications* 13.1 (2022), pp. 1–13.

[12] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural Architecture Search: A Survey." In: *The Journal of Machine Learning Research* 20.1 (2019), pp. 1997–2017.

[13] Eyrun Eyjolfsdottir et al. "Detecting Social Actions of Fruit Flies." In: *European Conference on Computer Vision*. Springer. 2014, pp. 772–787.

[14] Joseph Garner. *Mouse Ethogram – Stanford School of Medicine*. en-US. URL: `https://mousebehavior.org/` (visited on 11/04/2022).

[15] Noah D. Goodman, Tomer D. Ullman, and Joshua B. Tenenbaum. "Learning a Theory of Causality." eng. In: *Psychological Review* 118.1 (Jan. 2011), pp. 110–119. ISSN: 1939-1471. DOI: `10.1037/a0021336`.

[16] Nastacia L. Goodwin et al. "Toward the Explainability, Transparency, and Universality of Machine Learning for Behavioral Classification in Neuroscience." In: *Current Opinion in Neurobiology* 73 (2022), p. 102544.

[17] Sumit Gulwani. "Automating String Processing in Spreadsheets using Input-Output Examples." In: *PoPL'11, January 26-28, 2011, Austin, Texas, USA*. Jan. 2011.

[18] Minghao Guo et al. "Data-Efficient Graph Grammar Learning for Molecular Generation." In: *International Conference on Learning Representations*. 2021.

[19] Jon Kleinberg et al. "Human Decisions and Machine Predictions." In: *The Quarterly Journal of Economics* 133.1 (2018), pp. 237–293.

[20] Pang Wei Koh et al. "WILDS: A Benchmark of In-the-Wild Distribution Shifts." In: *International Conference on Machine Learning*. PMLR. 2021, pp. 5637–5664.

[21] Ali Sinan Koksal et al. "Synthesis of Biological Models from Mutation Experiments." In: *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 2013, pp. 469–482.

[22] Tessa A. Lau and Daniel S. Weld. "Programming by Demonstration: An Inductive Learning Formulation." In: *Proceedings of the 4th International Conference on Intelligent User Interfaces*. 1998, pp. 145–152.

[23] Xubo Leng et al. "Quantifying Influence of Human Choice on the Automated Detection of Drosophila Behavior by a Supervised Machine Learning Algorithm." In: *PLoS ONE* 15.12 (Dec. 2020), e0241696. ISSN: 1932-6203. DOI: `10.1371/journal.pone.0241696`. (Visited on 09/20/2022).

[24] Alexander Mathis et al. "DeepLabCut: Markerless Pose Estimation of User-Defined Body Parts with Deep Learning." In: *Nature Neuroscience* (2018). URL: `https://www.nature.com/articles/s41593-018-0209-y`.

[25] Thomas McGrath et al. "Acquisition of Chess Knowledge in AlphaZero." In: *arXiv preprint arXiv:2111.09259* (2021).

[26] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[27] Talmo D. Pereira, Joshua W. Shaevitz, and Mala Murthy. "Quantifying Behavior to Understand the Brain." In: *Nature Neuroscience* 23.12 (2020), pp. 1537–1549.

[28] Talmo D. Pereira et al. "SLEAP: A Deep Learning System for Multi-Animal Pose Tracking." In: *Nature Methods* 19.4 (2022), pp. 486–495.

[29] Cynthia Rudin. "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead." In: *Nature Machine Intelligence* 1.5 (2019), pp. 206–215.

[30] Nicolae Sapoval et al. "Current Progress and Open Challenges for Applying Deep Learning Across the Biosciences." In: *Nature Communications* 13.1 (2022), pp. 1–12.

[31] Cristina Segalin, Jalani Williams, Tomomi Karigo, May Hui, Moriel Zelikowsky, Jennifer J. Sun, Pietro Perona, David J. Anderson, and Ann Kennedy. "The Mouse Action Recognition System (MARS) Software Pipeline for Automated Analysis of Social Behaviors in Mice." In: *eLife* 10 (2021), e63720.

[32] Ameesh Shah*, Eric Zhan*, Jennifer J. Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. "Learning Differentiable Programs with Admissible Neural Heuristics." In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 4940–4952. URL: `https://arxiv.org/pdf/2007.12101.pdf`.

[33] Jennifer J. Sun, Ann Kennedy, Eric Zhan, David J. Anderson, Yisong Yue, and Pietro Perona. "Task Programming: Learning Data Efficient Behavior Representations." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 2876–2885. URL: `https://arxiv.org/pdf/2011.13917.pdf`.

[34] Jennifer J. Sun*, Serim Ryou*, Roni H. Goldshmid, Brandon Weissbourd, John O. Dabiri, David J. Anderson, Ann Kennedy, Yisong Yue, and Pietro Perona. "Self-Supervised Keypoint Discovery in Behavioral Videos." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 2171–2180. URL: `https://arxiv.org/pdf/2112.05121.pdf`.

[35] Megan Tjandrasuwita, Jennifer J. Sun, Ann Kennedy, Swarat Chaudhuri, and Yisong Yue. "Interpreting Expert Annotation Differences in Animal Behavior." In: *CV4Animals Workshop at the Conference on Computer Vision and Pattern Recognition (CVPR)* (2021). URL: `https://arxiv.org/pdf/2106.06114.pdf`.

[36] Albert Tseng, Jennifer J. Sun, and Yisong Yue. "Automatic Synthesis of Diverse Weak Supervision Sources for Behavior Analysis." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 2211–2220. URL: `https://arxiv.org/pdf/2111.15186.pdf`.

[37] Berk Ustun and Cynthia Rudin. “Optimized Risk Scores.” In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2017, pp. 1125–1134.

[38] Lazar Valkov et al. “Houdini: Lifelong Learning as Program Synthesis.” In: *Advances in neural information processing systems*. 2018.

[39] Abhinav Verma et al. “Programmatically interpretable reinforcement learning.” In: *International Conference on Machine Learning*. 2018.

[40] Catherine Wong et al. “Leveraging Language to Learn Program Abstractions and Search Heuristics.” In: *International Conference on Machine Learning*. PMLR. 2021, pp. 11193–11204.

[41] Eric Zhan*, Jennifer J. Sun*, Ann Kennedy, Yisong Yue, and Swarat Chaudhuri. “Unsupervised Learning of Neurosymbolic Encoders.” In: *Transactions on Machine Learning Research* (2022). URL: `https://arxiv.org/pdf/2107.13132.pdf`.

Part IV

Conclusion

Chapter 13

DISCUSSION AND FUTURE WORK

The development of AI for scientists requires a collaborative process between ML researchers and labs with diverse domain knowledge, experimental setups, and data. Through this process, my research lays the groundwork towards a general scientist-in-the-loop framework that learns from expert knowledge and experimental data across domains. This framework has the potential to unify analysis efforts and share insights broadly across science. My current work has began to study how workflows in fields such as behavioral neuroscience motivates the design of AI systems, as well as how these systems could benefit the workflow of scientists. Many open questions still remain: What is the most efficient way to encode domain knowledge into AI systems? How to build benchmarks and tools that are broadly applicable in the sciences? How to close the development-deployment loop between AI and science research communities. Towards this, future directions to explore include:

Enabling richer scientist-AI interactions. New methods and algorithms motivates new modes of interaction. Current developments has already led to multiple ways for scientists to interact with AI systems, such as through data annotation or task programming as we discussed earlier in this thesis. However, in these cases, scientists generally define everything about the task, and the model is simply reproducing human observations and annotations. Neurosymbolic models offer another layer of interaction with users through interpretability. Scientists have the potential to interpret these model directly to gain more insight into the phenomenon they are studying. For example, we have found that compared to neural networks, neurosymbolic methods result in qualitatively more interpretable features and patterns for studying mice behavior [4]. In addition, despite being much simpler and having fewer parameters, we found that learned programs performed comparably to the neural network in terms of accuracy.

The development of large language models (LLMs) and foundation models (FMs) has opened up new avenues for accelerating scientist-AI collaborations. These models can learn from vast amounts of data and generate insights that would be difficult or impossible for humans to obtain on their own. However, there is remains a gap between the general knowledge encoded by LLMs and the more domain-specific knowledge that scientists have in their experiments. The challenges are that scientists study diverse data modalities, such as infrared recordings, fluorescent imaging, or neural recordings of different parts of the brain, and often rare phenomenon where we do not have a lot of existing data. This makes it difficult to deploy LLMs and FMs out-of-the-box in these domain-specific experimental setups. There are exciting open questions on the extent to which LLMs can be deployed in domain-specific experimental setups and how much data we would need to fine-tune these models on new experiments. Our goal is to develop new methods to bridge the gap between general and domain-specific knowledge, so that scientists can leverage the power of LLMs and FMs to accelerate their research.

Expanding neurosymbolic modeling to new domains. The methods discussed in this thesis have been primarily applied to the modeling of human and animal behavior. However, these methods are also applicable to other types of spatiotemporal data, such as electrocardiogram (ECG) data. As an early exploration, we are working with cardiologists to learn interpretable neurosymbolic programs from ECG data to study different heart conditions. Our approach is to first define a space of programs by identifying interpretable ECG features. We then use learned temporal filters and compositions to synthesize these programs, using similar learning algorithms to those developed earlier for behavior.

Across scientific domains, knowledge is encoded in many forms, including mathematical equations, natural language, and knowledge graphs. We would like to build neurosymbolic methods that enable AI systems to ingest different types of knowledge. For example, a knowledge graph could be used to encode the relationships between genes in biomedical data analysis. However, these organized forms of knowledge do not exist everywhere. For example, there is no knowledge graph that comprehensively encodes prior results and theories in behavioral neuroscience. This is a challenge because it can be difficult for scientists to query for relevant knowledge or design new experiments. Therefore, in addition to developing new methods, we also need to work with the scientific community to study new ways to encode prior knowledge.

Building AI for the scientific process. The scientific process is an iterative process that involves hypothesis design, data collection, data analysis, interpretation, and communication to the community. We focused our discussion in this thesis on data analysis, but AI has the potential to impact all steps in the scientific process, from hypothesis design to sharing results. My goal is to build towards an AI ecosystem that can help scientists end-to-end through the scientific process. This system would need to be able to work with diverse data sources, collaborate with scientists to interpret results, and integrate new scientific advances.

Ultimately, for ML to achieve its full impact in science, we need to close the loop between model development and evolving real-world challenges from scientists. Currently, many benchmarks in ML and tools in science are focused on a single analysis task or domain but many computational challenges are shared more broadly [1, 3]. We need broad and joint efforts from the community, in order to develop diverse cross-domain datasets and benchmarks, as well as build cross-domain tools to translate CVML models to real-world use cases. Tackling these challenges does not only require new ML methods, but also requires community effort to realize the potential of ML in the sciences. I will continue to work with researchers across communities and build towards a unified scientist-in-the-loop system that enables sharing of data and insights for all scientists.