Understanding and Improving Reliability of Inference Dynamics in Deep Neural Networks

Thesis by
Yujia Huang

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

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2024
Defended January 8, 2024

Yujia Huang
ORCID: 0000-0001-7667-8342

ACKNOWLEDGEMENTS

Foremost, I extend my deepest gratitude to my advisor, Yisong Yue. You embraced me with trust and welcomed me into the vibrant Yue Crew, granting me the freedom to pursue my passions in research. You always keep our interests in mind and thrive to provide the best resources for us. I am profoundly grateful for the respect and equality you extend to all students, fostering an environment where I could freely express questions, concerns, and even failures. Your thoughtfulness in seeking efficient solutions is deeply appreciated. I am equally grateful for the inclusive and vibrant environment you've fostered in the lab, where research and enjoyment of life find a harmonious balance.

My heartfelt thanks also go to my committee members, Katie Bouman, Adam Wierman, Georgia Gkioxari, and Yuanyuan Shi. Your valuable feedback and guidance have been instrumental in refining my research.

To all my collaborators during my PhD journey — Huan Zhang, Yuanyuan Shi, Ivan Dario Jimenez Rodriguez, Sageev Oore, Adishree Ghatare, Yuanzhe Liu, Ziniu Hu, Qinsheng Zhang, Chandramouli Sastry, James Gornet, Sihui Dai, Tan Nguyen, Zhiding Yu, Weili Nie — each of you has enriched my experience immeasurably. Collaborating with you has been a privilege and I have learned so much from each of you.

To all my friends, who made my time at Caltech colorful. To the Yue Crew, for creating a lab atmosphere filled with warmth and fun — our happy hours will always be cherished. To Berthy Feng, Zongyi Li, Jiawei Zhao, Sahin Lale, Rafal Kocielnik, Cheng Shen, Yuanyuan Shi, Junmin Wu, Anqi Liu, Quanying Liu, Haiyan Wu and many others — thank you for the precious memories we've created over the years.

I am grateful to Laura Flower Kim and Daniel Yoder from ISP for your kindness and efficiency, making my journey as an international student seamless and enjoyable.

Finally, my deepest appreciation goes to my parents, Xiangming Huang and Xin Cheng, and my grandparents Jingsheng Huang and Shuzhen Xing, for their unconditional love and support from the very beginning. To my boyfriend, Fengyu Zhou, who brings me immense joy and offers patience and support when I need it most. My family is the wellspring of my inspiration and strength. This thesis belongs to all of you!

ABSTRACT

Reliability is a crucial aspect for the successful deployment of deep learning systems across various domains. In generative modeling, it is essential to create content that adheres to specific rules. In the field of control, ensuring that robots operate safely without falling or entering hazardous areas is paramount. Similarly, in visual perception, the robustness of perception results against perturbations are vital.

In this thesis, we explore the reliability of inference dynamics in deep neural networks such as ResNet, neural Ordinary Differential Equations (ODEs), and diffusion models. We begin by examining the inference dynamics in standard networks with a discrete sequence of hidden layers, applying self-consistency and local Lipschitz bounds to enhance robustness against input perturbations. Our exploration then extends to neural ODEs, where the neural network specifies a vector field that continuously transforms the state. We employ forward invariance to achieve robustness, marking the first instance of training neural ODE policies with non-vacuous certified guarantees. The focus shifts next to diffusion models and their inference processes, particularly in adhering to symbolic constraints. For this, we introduce a novel sampling algorithm inspired by stochastic control principles. This algorithm not only guides these models in generating rule-specific content but also sets a new benchmark in symbolic music generation. Our work offers a cohesive understanding of inference dynamics in various deep learning architectures and propose new algorithms to significantly improve their reliability.

PUBLISHED CONTENT AND CONTRIBUTIONS

Yujia Huang, Adishree Ghatare, Yuanzhe Liu, Ziniu Hu, Qinsheng Zhang, Chandramouli Sastry, Siddharth Gururani, Sageev Oore, and Yisong Yue (2024). "Symbolic Music Generation with Non-Differentiable Rule Guided Diffusion". In: *arXiv preprint arXiv:2402.14285*. URL: https://arxiv.org/abs/2402.14285.

Y.H. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and participated in the writing of the manuscript.

Yujia Huang, Ivan Dario Jimenez Rodriguez, Huan Zhang, Yuanyuan Shi, and Yisong Yue (2023). "FI-ODE: Certified and Robust Forward Invariance in Neural ODEs". In: *arXiv preprint arXiv:2210.16940*. URL: https://arxiv.org/abs/2210.16940.

Y.H. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and participated in the writing of the manuscript.

Yujia Huang, Huan Zhang, Yuanyuan Shi, J. Zico Kolter, and Anima Anandkumar (2021). "Training Certifiably Robust Neural Networks with Efficient Local Lipschitz Bounds". In: *Advances in Neural Information Processing Systems* 34, pp. 22745–22757. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/c055dcc749c2632fd4dd806301f05ba6-Paper.pdf.

Y.H. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and participated in the writing of the manuscript.

Yujia Huang, James Gornet, Sihui Dai, Zhiding Yu, Tan Nguyen, Doris Tsao, and Anima Anandkumar (2020). "Neural Networks with Recurrent Generative Feedback". In: *Advances in Neural Information Processing Systems* 33, pp. 535–545. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/0660895c22f8a14eb039bfb9beb0778f-Paper.pdf.

Y.H. participated in the conception of the project, formulated and implemented the method, conducted experiments and analyzed results, and participated in the writing of the manuscript.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Number *Page*

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 The Reliability Issue in Deep Learning

Deep neural networks have been widely applied in many real-world problems. Their proficiency in learning intricate patterns and executing complex predictions or decisions is remarkable. Yet, a persistent concern remains their reliability.

In this thesis, we explore the reliability issue within three distinct domains: generative modeling, control, and visual perception.

Generative Modeling. Diffusion models, while powerful, sometimes generate images that deviate from specific text prompts, particularly in adhering to rule-based specifications such as binding attributes to objects (Liu et al., 2023). In this scenario, reliability is being consistent with the rules used to guide the diffusion. This thesis introduces novel algorithms designed to steer diffusion models towards generating content that complies with these rule specifications in a flexible, plug-and-play manner.

Control. In control systems, reliability means the ability of controllers to maintain safety guarantees under perturbations. For instance, ensuring a Segway robot remains upright on uneven terrain, or a drone maintains steady flight in turbulent winds. Although the field of robust control (Zhou and Doyle, 1998) have succeeded in designing controllers that are provably robust even under the worst conditions, the resulting policies tend to be often linear, constraining their effectiveness. Our work proposes innovative methods for training nonlinear control policies parameterized by neural networks, with verifiable robustness guarantees.

Visual Perception. Despite the significant achievements of deep neural networks in visual perception, they are known to be vulnerable to adversarial attacks (Szegedy et al., 2014). For example, imperceptible alterations to an image can mislead a model into incorrect classifications. Reliability in this sense means that the visual perception system still makes correct predictions under input perturbations. This thesis presents strategies to bolster the adversarial robustness of image classifiers, both empirically and through provable methods.

To enhance the reliability of deep learning systems, a fundamental understanding

of the inherent unreliability of neural networks is essential. This is challenging because neural networks are comprised of layers of highly over-parameterized neurons. This over-parameterization contributes to the complexity of the networks, making it particularly challenging to dissect and understand the behaviors of such high-dimensional computational systems.

This thesis aims to deepen our understanding and enhance the reliability of deep learning systems through the lens of **inference dynamics**. We will delve into the inference dynamics of several prominent deep learning models in the following section, seeking breakthroughs to improve reliability through innovative learning and inference algorithms.

1.2 Inference Dynamics of Deep Learning Systems

In this section, we examine the inference dynamics of three prominent deep learning models: feedforward neural networks, neural ordinary differential equations (neural ODEs), and diffusion models.

Feedforward Neural Networks. Feedforward neural networks are foundational building blocks of deep learning systems. In a feedforward neural network, the information always flow in one direction from input to the output. Due to this one-way flow, these networks lack the ability to revise outputs post-generation, posing a challenge in cases of erroneous predictions. In Chapter 2, we introduce novel algorithms to that empower these networks to refine predictions during testing time. Figure 1.1 shows the composite layers of feedforward deep neural networks.

More formally, considering $\boldsymbol{\eta}_n \in \mathbb{R}^d$ as the hidden states of layer n, the states of the subsequent layer are determined through:

$$\boldsymbol{\eta}_n = \boldsymbol{f}(\boldsymbol{\eta}_{n-1}, \boldsymbol{\theta}_n) \tag{1.1}$$

where n ranges from 1 to N, and $\boldsymbol{\theta}_n$ represents the model parameters at layer n. A fundamental architecture in deep neural networks is the residual network (He et al., 2016), which incorporates skip connections between layers to facilitate network optimization. The update rule in residual neural networks is:

$$\boldsymbol{\eta}_n = \boldsymbol{\eta}_{n-1} + \boldsymbol{f}(\boldsymbol{\eta}_{n-1}, \boldsymbol{\theta}_n) \tag{1.2}$$

While this sequence of transformations ($\boldsymbol{f}$) endows the feedforward network with the capability to approximate complex functions, it can also amplify input noise. This amplification can transform minor noise into significant errors in output. This

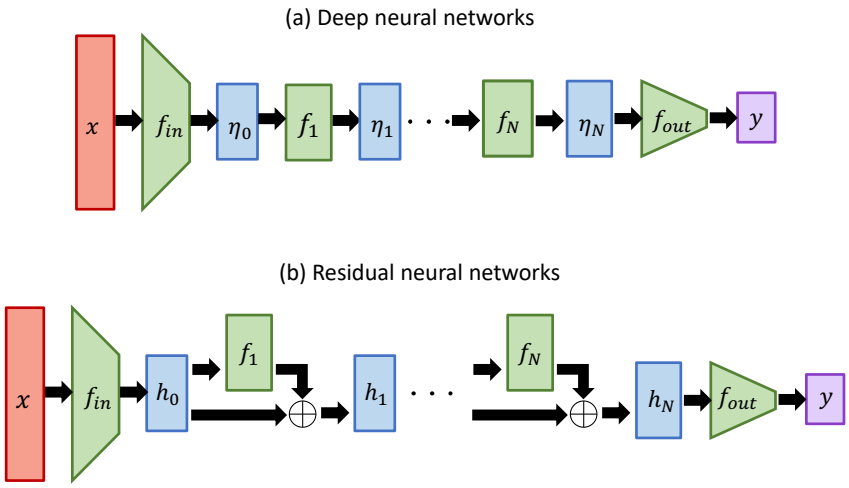


Figure 1.1: **Deep Learning Architectures.** (a) Composite layers in a deep neural network. x, y are the input and output respectively. f stands for per-layer functions, and η stands for the hidden states. (b) Illustration of a ResNet architecture. There are skipping connections between layers.

amplification can transform minor noise in input into significant errors in output. In Chapter 3, we propose efficient bounds to mitigate noise amplification at each layer, thereby offering certified robustness guarantees for the network.

Neural ODEs. In contrast to the discrete layer structure of feedforward networks, neural ODEs model the continuous dynamics of hidden states using ordinary differential equations. The output is derived as the numerical solution of the ODE, starting from an initial condition to a terminal time T. The derivative of the hidden states is formally expressed as:

$$\frac{d\boldsymbol{\eta}_t}{dt} = \boldsymbol{f}(\boldsymbol{\eta}_t, t, \boldsymbol{\theta}) \tag{1.3}$$

with $\boldsymbol{\eta}_T$ being the solution to the ODE given the initial condition $\boldsymbol{\eta}_0$.

Neural ODEs are particularly useful in applications involving continuous dynamics, such as generative modeling (Chen et al., 2018), time series analysis (Rubanova et al., 2019), and control (Böttcher and Asikis, 2022). They also provide a novel lens to understand the inference dynamics of residual networks (He et al., 2016), viewed as an Euler discretization of a continuous transformation (Lu et al., 2018).

The continuous nature of neural ODEs enables incorporating control-theoretic concepts such as forward invariance to boost their reliability. Forward invariance guarantees that NODE trajectories never leave a specified set, which can be translated into various robust safety guarantees. In Chapter 4, we introduce a general framework for training certifiably robust forward invariant neural ODEs.

Diffusion Models. Diffusion models, as detailed in key works (Sohl-Dickstein et al., 2015; Ho et al., 2020), represent a powerful class of generative models that create data by reversing a diffusion process. A continuous perspective of this process is provided by the use of stochastic differential equations (SDEs) to describe both the forward diffusion and its reverse-time counterpart (Song et al., 2021). This reverse-time SDE relies on a neural network approximated score function, with sample generation achieved through numerical SDE solvers. Formally, the generative process is defined as:

$$d\boldsymbol{\eta}_t = \boldsymbol{f}(\boldsymbol{\eta}_t, t, s_{\boldsymbol{\theta}})dt + g(t)d\bar{\boldsymbol{w}} \tag{1.4}$$

where dt is an infinitesimal negative time step and $\bar{\boldsymbol{w}}$ is a standard reverse-time Wiener process. $\boldsymbol{f}$ represents the transformation leveraging the neural network-parameterized score function $s_{\boldsymbol{\theta}}$.

In Chapter 5, we study how to guide the generative process of diffusion models to conform to rule specifications via stochastic control. This involves integrating the optimal control $\boldsymbol{u}_t$ into the drift term of the SDE, resulting in $\boldsymbol{f}(\boldsymbol{\eta}t, t, s\boldsymbol{\theta}, \boldsymbol{u}_t)$. Solving this modified SDE facilitates the production of samples that adhere to predefined rules.

Visualizing the Hidden States. Figure 1.2 presents a visualization of the one-dimensional hidden states corresponding to the inference dynamics previously discussed. In the case of ResNet, the x-axis represents the number of layers, and the updates are defined only at discrete layers. For neural ODEs, the x-axis denotes time, with the trajectory being obtained by integrating an ODE over this temporal dimension. In the context of neural SDEs (the function class of diffusion models), the trajectory is influenced at each step by a diffusion term, adding noise to the process.

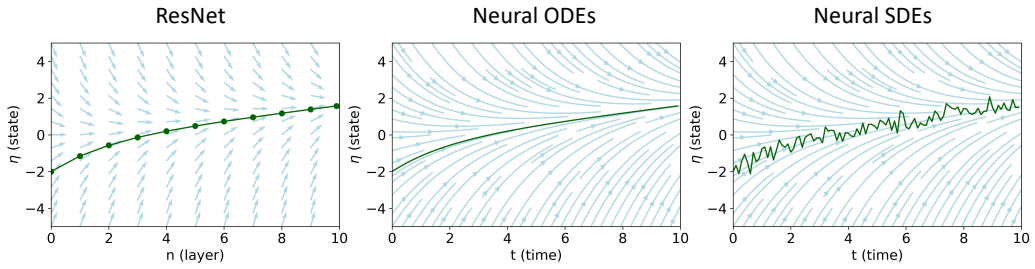


Figure 1.2: **Visual representation of the one-dimensional hidden states for ResNet, neural ODE, and neural SDE.**

1.3 Thesis Structure and Contributions

This thesis delves into the inference dynamics of deep learning systems, showcasing enhanced reliability across several applications such as generative modeling, control, and image classification.

Chapter 2 & Chapter 3 – Enhancing Neural Network Robustness

Chapters 2 and 3 are dedicated to improving the adversarial robustness of standard neural networks. In Chapter 2, we present a novel approach, Convolutional Neural Networks with Feedback (CNN-F). This method integrates generative feedback with latent variables into existing feedforward architectures, enabling consistent predictions through iterative maximum a posteriori inference. CNN-F demonstrates significantly enhanced empirical adversarial robustness compared to traditional feedforward CNNs.

Chapter 3 shifts focus to providing neural networks with a certified robustness guarantee. We introduce a trainable and efficient local Lipschitz upper bound, factoring in the interactions between activation functions (like ReLU) and weight matrices. This approach consistently surpasses current leading methods in terms of both clean and certified accuracy on established benchmarks.

Chapter 4 – Control-Theoretic Tools in Neural ODEs

In Chapter 4, we explore the application of control theory in neural ODEs (NODEs) to establish robust safety guarantees. Our interest centers on a control theory concept known as forward invariance, used to ensure a dynamical system remains within a certain state set indefinitely, even under perturbations. This chapter demonstrates certified robustness in both nonlinear Neural ODE control and image classification, marking a step towards certifying complex NODEs across various domains.

Chapter 5 – Guiding Diffusion Models with Symbolic Constraints

Chapter 5 addresses the challenge of directing diffusion models to produce content that adheres to specific symbolic constraints. In fields like symbolic music generation, creating rules that define desired output characteristics (such as note density or chord progressions) is relatively straightforward. However, incorporating these rules during the training phase presents substantial computational hurdles. Recognizing the need for a more efficient approach, we introduce Stochastic Control Guidance (SCG), a novel method offering plug-and-play guidance for non-differentiable rules. This represents the first instance of such an approach being successfully applied.

References

Lucas Böttcher and Thomas Asikis (2022). "Near-optimal Control of Dynamical Systems with Neural Ordinary Differential Equations". In: *Machine Learning: Science and Technology* 3.4, p. 045004.

Ricky T.Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud (2018). "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Jonathan Ho, Ajay Jain, and Pieter Abbeel (2020). "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems* 33, pp. 6840–6851.

Qihao Liu, Adam Kortylewski, Yutong Bai, Song Bai, and Alan Yuille (2023). "Discovering Failure Modes of Text-guided Diffusion Models via Adversarial Search". In: *arXiv preprint arXiv:2306.00974*.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong (2018). "Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations". In: *International Conference on Machine Learning*, pp. 3276–3285.

Yulia Rubanova, Ricky T.Q. Chen, and David K Duvenaud (2019). "Latent Ordinary Differential Equations for Irregularly-sampled Time Series". In: *Advances in Neural Information Processing Systems* 32.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli (2015). "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *International Conference on Machine Learning*, pp. 2256–2265.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole (2021). "Score-based Generative Modeling Through Stochastic Differential Equations". In: *International Conference on Learning Representations*.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2014). "Intriguing Properties of Neural Networks". In: *International Conference on Learning Representations*.

Kemin Zhou and John Comstock Doyle (1998). *Essentials of Robust Control*. Vol. 104. Prentice Hall Upper Saddle River, NJ.

Chapter 2

INTRODUCING RECURRENT GENERATIVE FEEDBACK TO FEEDFORWARD NEURAL NETWORKS

Yujia Huang, James Gornet, Sihui Dai, Zhiding Yu, Tan Nguyen, Doris Tsao, and Anima Anandkumar (2020). "Neural Networks with Recurrent Generative Feedback". In: *Advances in Neural Information Processing Systems* 33, pp. 535–545. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/0660895c22f8a14eb039bfb9beb0778f-Paper.pdf.

2.1 Introduction

Conventional deep neural networks (DNNs) often contain many layers of feedforward connections. With the ever-growing network capacities and representation abilities, they have achieved great success. For example, recent convolutional neural networks (CNNs) have impressive accuracy on large scale image classification benchmarks (Szegedy et al., 2016). However, current CNN models also have significant limitations. For instance, they can suffer significant performance drop from corruptions which barely influence human recognition (Dodge and Karam, 2017). Studies also show that CNNs can be misled by imperceptible noise known as adversarial attacks (Szegedy et al., 2014).

To address the weaknesses of CNNs, we can take inspiration from of how human visual recognition works, and incorporate certain mechanisms into the CNN design.

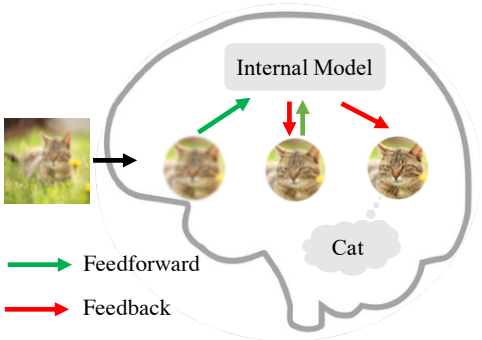


Figure 2.1: **An intuitive illustration of recurrent generative feedback in human visual perception system.**

While human visual cortex has hierarchical feedforward connections, backward connections from higher level to lower level cortical areas are something that current artificial networks are lacking (Felleman and Essen, 1991). Studies suggest these backward connections carry out top-down processing which improves the representation of sensory input (Kok et al., 2012). In addition, evidence suggests recurrent feedback in the human visual cortex is crucial for robust object recognition. For example, humans require recurrent feedback to recognize challenging images (Kar et al., 2019). Obfuscated images can fool humans without recurrent feedback (Elsayed et al., 2018). Figure 2.1 shows an intuitive example of recovering a sharpened cat from a blurry cat and achieving consistent predictions after several iterations.

Computational neuroscientists speculate that Bayesian inference models human perception (Knill and Richards, 1996). One specific formulation of predictive coding assumes Gaussian distributions on all variables and performs hierarchical Bayesian inference using recurrent, generative feedback pathways (Rao and Ballard, 1999). The feedback pathways encode predictions of lower level inputs, and the residual errors are used recurrently to update the predictions. In this paper, we extend the principle of predictive coding to explicitly incorporate Bayesian inference in neural networks via generative feedback connections. Specifically, we adopt a recently proposed model, named the Deconvolutional Generative Model (DGM) (Nguyen et al., 2018), as the generative feedback. The DGM introduces hierarchical latent variables to capture variation in images, and generates images from a coarse to fine detail using deconvolutional operations.

Our contributions are as follows:

Self-consistency. We introduce generative feedback to neural networks and propose the self-consistency formulation for robust perception. Our internal model of the world reaches a self-consistent representation of an external stimulus. Intuitively, self-consistency says that given any two elements of label, image and auxillary information, we should be able to infer the other one. Mathematically, we use a generative model to describe the joint distribution of labels, latent variables and input image features. If the MAP estimate of each one of them are consistent with the other two, we call a label, a set of latent variables and image features to be self-consistent (Figure 2.4).

CNN with Feedback (CNN-F). We incorporate generative recurrent feedback modeled by the DGM into CNN and term this model as CNN-F. We show that Bayesian inference in the DGM is achieved by CNN with adaptive nonlinear operators (Figure

2.2). We impose self-consistency in the CNN-F by iterative inference and online update. Computationally, this process is done by propagating along the feedforward and feedback pathways in the CNN-F iteratively (Figure 2.3).

Adversarial Robustness. We show that the recurrent generative feedback in CNN-F promotes robustness and visualizes the behavior of CNN-F over iterations. We find that more iterations are needed to reach self-consistent prediction for images with larger perturbation, indicating that recurrent feedback is crucial for recognizing challenging images. When combined with adversarial training, CNN-F further improves adversarial robustness of CNN on both Fashion-MNIST and CIFAR-10 datasets. Code is available at `https://github.com/yjhuangcd/CNNF`.

2.2 Self-Consistency and Recurrent Generative Feedback

In this section, we first formally define self-consistency. Then we give a specific form of generative feedback in CNN and impose self-consistency on it. We term this model as CNN-F. Finally we show the training and testing procedure in CNN-F. Throughout, we use the following notations:

Let $x \in \mathbb{R}^n$ be the input of a network and $y \in \mathbb{R}^K$ be the output. In image classification, x is image and $y = (y^{(1)}, \ldots, y^{(K)})$ is one-hot encoded label. K is the total number of classes. K is usually much less than n. We use L to denote the total number of network layers, and index the input layer to the feedforward network as layer 0. Let $h \in \mathbb{R}^m$ be encoded feature of x at layer k of the feedforward pathway. Feedforward pathway computes feature map $f(\ell)$ from layer 0 to layer

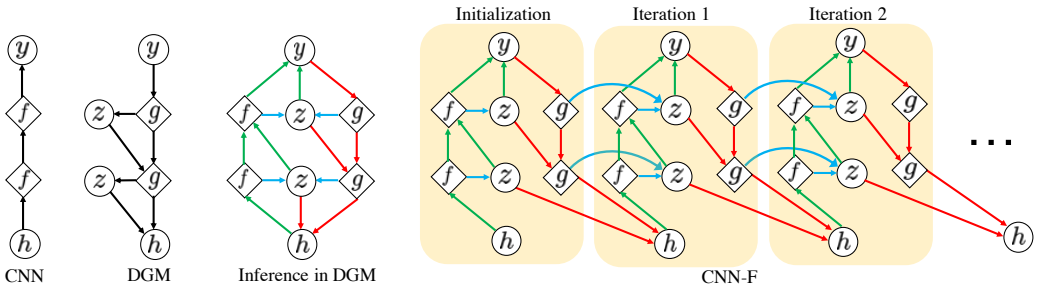


Figure 2.2: **Inference in CNN-F. Left: CNN, Graphical model for the DGM and the inference network for the DGM.** We use the DGM to as the generative model for the joint distribution of image features h, labels y and latent variables z. MAP inference for h, y and z is denoted in red, green and blue, respectively. f and g denotes feedforward features and feedback features, respectively. **Right: CNN with feedback (CNN-F)**. CNN-F performs alternating MAP inference via recurrent feedforward and feedback pathways to enforce self-consistency.

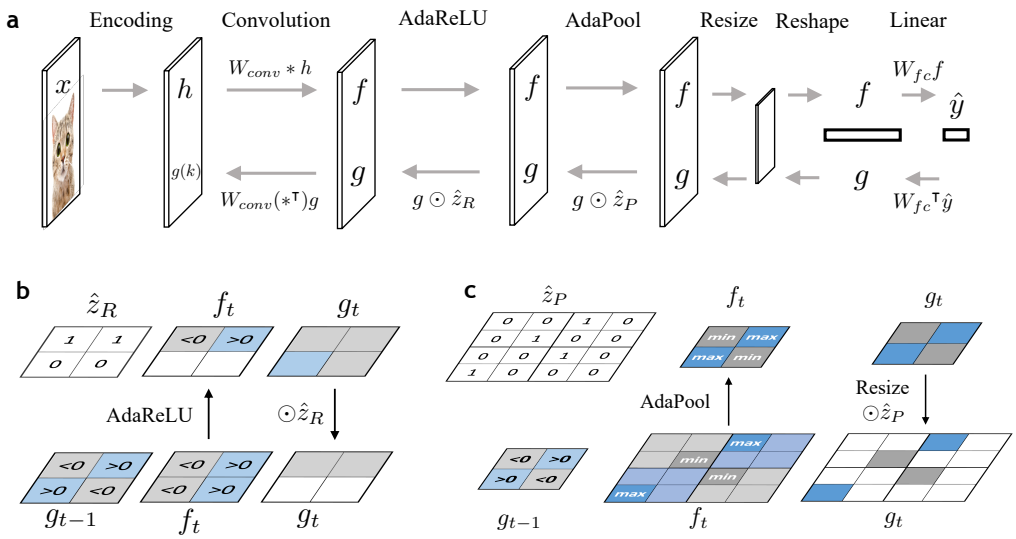


Figure 2.3: **Feedforward and feedback pathway in CNN-F.** a) $\hat{y}$ and $\hat{z}$ are computed by the feedforward pathway and $\hat{h}$ is computed from the feedback pathway. b) Illustration of the AdaReLU operator. c) Illustration of the AdaPool operator.

L, and feedback pathway generates $g(\ell)$ from layer L to k. $g(\ell)$ and $f(\ell)$ have the same dimensions. To generate h from y, we introduce latent variables for each layer of CNN. Let $z(\ell) \in \mathbb{R}^{C \times H \times W}$ be latent variables at layer ℓ, where C, H, W are the number of channels, height and width for the corresponding feature map. Finally, $p(h, y, z; \theta)$ denotes the joint distribution parameterized by θ, where θ includes the weight W and bias term b of convolution and fully connected layers. We use $\hat{h}$, $\hat{y}$ and $\hat{z}$ to denote the MAP estimates of h, y, z conditioning on the other two variables.

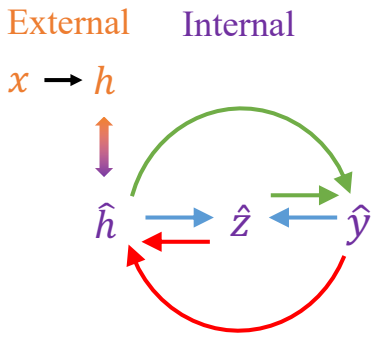


Figure 2.4: **Self-consistency among $\hat{h}, \hat{z}, \hat{y}$ and consistency between $\hat{h}$ and h.**

Generative Feedback and Self-consistency

Human brain and neural networks are similar in having a hierarchical structure. In human visual perception, external stimuli are first preprocessed by lateral geniculate nucleus (LGN) and then sent to be processed by V1, V2, V4 and Inferior Temporal (IT) cortex in the ventral cortical visual system. Conventional NN use feedforward layers to model this process and learn a one-direction mapping from input to output. However, numerous studies suggest that in addition to the feedforward connections from V1 to IT, there are feedback connections among these cortical areas (Felleman and Essen, 1991).

Inspired by the Bayesian brain hypothesis and the predictive coding theory, we propose to add generative feedback connections to NN. Since h is usually of much higher dimension than y, we introduce latent variables z to account for the information loss in the feedforward process. We then propose to model the feedback connections as MAP estimation from an internal generative model that describes the joint distribution of h, z and y. Furthermore, we realize recurrent feedback by imposing self-consistency (Definition 2.2.1).

Definition 2.2.1. (Self-consistency) Given a joint distribution $p(h, y, z; \theta)$ parameterized by θ, $(\hat{h}, \hat{y}, \hat{z})$ are self-consistent if they satisfy the following constraints:

$$\hat{y} = \arg\max_{y} p(y|\hat{h}, \hat{z}), \qquad \hat{h} = \arg\max_{h} p(h|\hat{y}, \hat{z}), \qquad \hat{z} = \arg\max_{z} p(z|\hat{h}, \hat{y}). \tag{2.1}$$

In words, self-consistency means that MAP estimates from an internal generative model are consistent with each other. In addition to self-consistency, we also impose the consistency constraint between $\hat{h}$ and the external input features (Figure 2.4). We hypothesize that for *easy* images (familiar images to human, clean images in the training dataset for NN), the $\hat{y}$ from the first feedforward pass should automatically satisfy the self-consistent constraints. Therefore, feedback need not be triggered. For *challenging* images (unfamiliar images to human, unseen perturbed images for NN), recurrent feedback is needed to obtain self-consistent $(\hat{h}, \hat{y}, \hat{z})$ and to match $\hat{h}$ with h. Such recurrence resembles the dynamics in neural circuits (Kietzmann et al., 2019) and the extra effort to process challenging images (Kar et al., 2019).

Generative Feedback in CNN-F

CNN have been used to model the hierarchical structure of human retinatopic fields (Eickenberg et al., 2017; Horikawa and Kamitani, 2017), and have achieved

state-of-the-art performance in image classification. Therefore, we introduce generative feedback to CNN and impose self-consistency on it. We term the resulting model as CNN-F.

We choose to use the DGM (Nguyen et al., 2018) as generative feedback in the CNN-F. The DGM introduces hierarchical binary latent variables and generates images from coarse to fine details. The generation process in the DGM is shown in Figure 2.3 (a). First, y is sampled from the label distribution. Then each entry of $z(\ell)$ is sampled from a Bernoulli distribution parameterized by $g(\ell)$ and a bias term $b(\ell)$. $g(\ell)$ and $z(\ell)$ are then used to generate the layer below:

$$g(\ell - 1) = W(*^{\intercal})(\ell)(z(\ell) \odot g(\ell)). \tag{2.2}$$

In this paper, we assume $p(y)$ to be uniform, which is realistic under the balanced label scenario. We assume that h follows Gaussian distribution centered at $g(k)$ with standard deviation σ.

Recurrence in CNN-F

In this section, we show that self-consistent $(\hat{h}, \hat{y}, \hat{z})$ in the DGM can be obtained via alternately propagating along feedforward and feedback pathway in CNN-F.

Feedforward and Feedback Pathway in CNN-F. The feedback pathway in CNN-F takes the same form as the generation process in the DGM (Equation (2.2)). The feedforward pathway in CNN-F takes the same form as CNN except for the nonlinear operators. In conventional CNN, nonlinear operators are $\sigma_{\text{ReLU}}(f) = \max(f, 0)$ and $\sigma_{\text{MaxPool}}(f) = \max_{r \times r} f$, where r is the dimension of the pooling region in the feature map (typically equals to 2 or 3). In contrast, we use σ_{AdaReLU} (Equation 2.3) and σ_{AdaPool} (Equation 2.4) given in in the feedforward pathway of CNN-F. These operators adaptively choose how to activate the feedforward feature map based on the sign of the feedback feature map. The feedforward pathway computes $f(\ell)$ using the recursion $f(\ell) = W(\ell) * \sigma(f(\ell - 1))\} + b(\ell)$ [1].

$$\sigma_{\text{AdaReLU}}(f) = \begin{cases} \sigma_{\text{ReLU}}(f), & \text{if } g \geq 0 \\ \sigma_{\text{ReLU}}(-f), & \text{if } g < 0 \end{cases} \tag{2.3}$$

$$\sigma_{\text{AdaPool}}(f) = \begin{cases} \sigma_{\text{MaxPool}}(f), & \text{if } g \geq 0 \\ -\sigma_{\text{MaxPool}}(-f), & \text{if } g < 0 \end{cases} \tag{2.4}$$

[1] σ takes the form of σ_{AdaPool} or σ_{AdaReLU}.

MAP inference in the DGM. Given a joint distribution of h, y, z modeled by the DGM, we aim to show that we can make predictions using a CNN architecture following the Bayes rule (Theorem 2.2.1). To see this, first recall that generative classifiers learn a joint distribution $p(x, y)$ of input data x and their labels y, and make predictions by computing $p(y|x)$ using the Bayes rule. A well known example is the Gaussian Naive Bayes model (GNB). The GNB models $p(x, y)$ by $p(y)p(x|y)$, where y is Boolean variable following a Bernoulli distribution and $p(x|y)$ follows Gaussian distribution. It can be shown that $p(y|x)$ computed from GNB has the same parametric form as logistic regression.

Assumption 2.2.1. (Constancy assumption in the DGM).

- **A.** The generated image $g(k)$ at layer k of DGM satisfies $||g(k)||_2^2 = \text{const}$.
- **B.** Prior distribution on the label is a uniform distribution: $p(y) = \text{const}$.
- **C.** Normalization factor in $p(z|y)$ for each category is constant: $\sum_z e^{\eta(y,z)} = \text{const}$.

Remark. To meet Assumption 2.2.1.A, we can normalize $g(k)$ for all k. This results in a form similar to the instance normalization that is widely used in image stylization (Ulyanov et al., 2016). See Appendix A.1 for more detailed discussion. Assumption 2.2.1.B assumes that the label distribution is balanced. η in Assumption 2.2.1.C is used to parameterize $p(z|y)$. See Appendix A.1 for the detailed form.

Theorem 2.2.1. Under Assumption 2.2.1 and given a joint distribution $p(h, y, z)$ modeled by the DGM, $p(y|h, z)$ has the same parametric form as a CNN with σ_{AdaReLU} and σ_{AdaPool}.

Proof. Please refer to Appendix A.1. □

Remark. Theorem 2.2.1 says that DGM and CNN is a generative-discriminative pair in analogy to GNB and logistic regression.

We also find the form of MAP inference for image feature $\hat{h}$ and latent variables $\hat{z}$ in the DGM. Specifically, we use z_R and z_P to denote latent variables that are at a layer followed by $\mathrm{AdaReLU}$ and $\mathrm{AdaPool}$, respectively. $\mathbb{1}(\cdot)$ denotes indicator function.

Proposition 2.2.1 (MAP inference in the DGM)**.** Under Assumption 2.2.1, the following hold:

- **A.** Let h be the feature at layer k, then $\hat{h} = g(k)$.

B. MAP estimate of $z(\ell)$ conditioned on h, y and $\{z(j)\}_{j\neq\ell}$ in the DGM is:

$$\hat{z}_R(\ell) = \mathbb{1}(\sigma_{\text{AdaReLU}}(f(\ell)) \geq 0) \tag{2.5}$$

$$\hat{z}_P(\ell) = \mathbb{1}(g(\ell) \geq 0) \odot \arg\max_{r\times r}(f(\ell)) + \mathbb{1}(g(\ell) < 0) \odot \arg\min_{r\times r}(f(\ell)). \tag{2.6}$$

Proof. For part A, we have $\hat{h} = \arg\max_h p(h|\hat{y}, \hat{z}) = \arg\max_h p(h|g(k)) = g(k)$. The second equality is obtained because $g(k)$ is a deterministic function of $\hat{y}$ and $\hat{z}$. The third equality is obtained because $h \sim \mathcal{N}(g(k), \text{diag}(\sigma^2))$. For part B, please refer to Appendix A.1. □

Remark. Proposition 2.2.1.A show that $\hat{h}$ is the output of the generative feedback in the CNN-F. Proposition 2.2.1.B says that $\hat{z}_R = 1$ if the sign of the feedforward feature map matches with that of the feedback feature map. $\hat{z}_P = 1$ at locations that satisfy one of these two requirements: 1) the value in the feedback feature map is non-negative and it is the maximum value within the local pooling region or 2) the value in the feedback feature map is negative and it is the minimum value within the local pooling region. Using Proposition 2.2.1.B, we approximate $\{\hat{z}(\ell)\}_{\ell=1:L}$ by greedily finding the MAP estimate of $\hat{z}(\ell)$ conditioning on all other layers.

Iterative inference and online update in CNN-F. We find self-consistent $(\hat{h}, \hat{y}, \hat{z})$ by iterative inference and online update (Algorithm 1). In the initialization step, image x is first encoded to h by k convolutional layers. Then h passes through a standard CNN, and latent variables are initialized with conventional σ_{ReLU} and σ_{MaxPool}. The feedback generative network then uses $\hat{y}_0$ and $\{\hat{z}_0(\ell)\}_{\ell=k:L}$ to generate intermediate features $\{g_0(\ell)\}_{\ell=k:L}$, where the subscript denotes the number of iterations. In practice, we use logits instead of one-hot encoded label in the generative feedback to maintain uncertainty in each category. We use $g_0(k)$ as the input features for the first iteration. Starting from this iteration, we use σ_{AdaReLU} and σ_{AdaPool} instead of σ_{ReLU} and and σ_{MaxPool} in the feedforward pathway to infer $\hat{z}$ (Equation (2.5) and (2.6)). In practice, we find that instead of greedily replacing the input with generated features and starting a new inference iteration, online update eases the training and gives better robustness performance. The online update rule of CNN-F can be written as:

$$\hat{h}_{t+1} \leftarrow \hat{h}_t + \eta(g_{t+1}(k) - \hat{h}_t) \tag{2.7}$$

$$f_{t+1}(\ell) \leftarrow f_{t+1}(\ell) + \eta(g_t(\ell) - f_{t+1}(\ell)), \ell = k, \ldots, L \tag{2.8}$$

where η is the step size. Greedily replacement is a special case for the online update rule when $\eta = 1$.

Algorithm 1 Iterative inference and online update in CNN-F

Require: Input image x, number of encoding layers k, maximum number of iterations N

Encode image x to h_0 with k convolutional layers
Initialize $\{\hat{z}(\ell)\}_{\ell=k:L}$ by σ_{ReLU} and σ_{MaxPool} in the standard CNN
while $t < N$ **do**
    Feedback pathway: generate $g_t(k)$ using $\hat{y}_t$ and $\hat{z}_t(\ell)$, $\ell = k, \ldots, L$
    Feedforward pathway:
        Use $\hat{h}_{t+1}$ as the input (Equation (2.7))
        Update each feedforward layer using Equation (2.8)
        Predict $\hat{y}_{t+1}$ using the updated feedforward layers
end while
return $\hat{h}_N, \hat{y}_N, \hat{z}_N$

Training the CNN-F

During training, we have three goals: 1) train a generative model to model the data distribution, 2) train a generative classifier, and 3) enforce self-consistency in the model. We first approximate self-consistent $(\hat{h}, \hat{y}, \hat{z})$ and then update model parameters based on the losses listed in Table 2.1. All losses are computed for every iteration. Minimizing the reconstruction loss increases data likelihood given current estimates of label and latent variables $\log p(h|\hat{y}_t, \hat{z}_t)$ and enforces consistency between $\hat{h}_t$ and h. Minimizing the cross-entropy loss helps with the classification goal. In addition to reconstruction loss at the input layer, we also add reconstruction loss between intermediate feedback and feedforward feature maps. These intermediate losses helps stabilizing the gradients when training an iterative model like the CNN-F.

Table 2.1: Training losses in the CNN-F.

	Form	Purpose
Cross-entropy loss	$\log p(y \mid \hat{h}_t, \hat{z}_t; \theta)$	classification
Reconstruction loss	$\log p(h \mid \hat{y}_t, \hat{z}_t; \theta) = \|\|h - \hat{h}\|\|_2^2$.	generation, self-consistency
Intermediate reconstruction loss	$\|\|f_0(\ell) - g_t(\ell)\|\|_2^2$	stabilizing training

2.3 Experiment

Generative Feedback Promotes Robustness

As a sanity check, we train a CNN-F model with two convolution layers and one fully-connected layer on clean Fashion-MNIST images. We expect that CNN-F

reconstructs the perturbed inputs to their clean version and makes self-consistent predictions. To this end, we verify the hypothesis by evaluating adversarial robustness of CNN-F and visualizing the restored images over iterations.

Adversarial Robustness. Since CNN-F is an iterative model, we consider two attack methods: attacking the first or last output from the feedforward streams. We use "first" and "e2e" (short for end-to-end) to refer to the above two attack approaches, respectively. Due to the approximation of non-differentiable activation operators and the depth of the unrolled CNN-F, end-to-end attack is weaker than first attack (Appendix A.2). We report the adversarial accuracy against the stronger attack in Figure 2.5. We use the Fast Gradient Sign Attack Method (FGSM) (Goodfellow et al., 2015) Projected Gradient Descent (PGD) method to attack. For PGD attack, we generate adversarial samples within L_∞-norm constraint, and denote the maximum L_∞-norm between adversarial images and clean images as ϵ.

Figure 2.5 (a, b) shows that the CNN-F improves adversarial robustness of a CNN on Fashion-MNIST without access to adversarial images during training. The error bar shows standard deviation of 5 runs. Figure 2.5 (c) shows that training a CNN-F with more iterations improves robustness. Figure 2.5 (d) shows that the predictions are corrected over iterations during testing time for a CNN-F trained with 5 iterations. Furthermore, we see larger improvements for higher ϵ. This indicates that recurrent feedback is crucial for recognizing challenging images.

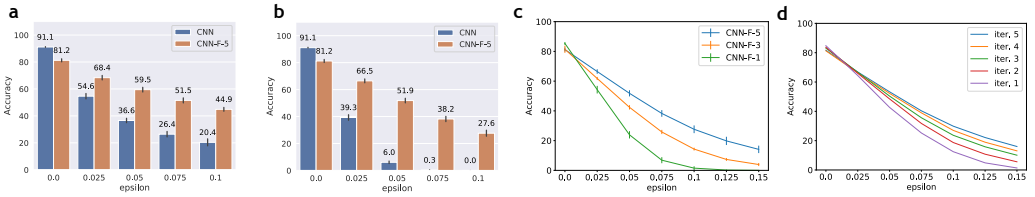


Figure 2.5: **Adversarial robustness of CNN-F with standard training on Fashion-MNIST.** CNN-F-k stands for CNN-F trained with k iterations. a) Attack with FGSM. b) Attack with PGD using 40 steps. c) Train with different number of iterations. Attack with PGD-40. d) Evaluate a trained CNN-F-5 model with various number of iterations against PGD-40 attack.

Image Restoration. Given that CNN-F models are robust to adversarial attacks, we examine the models' mechanism for robustness by visualizing how the generative feedback moves a perturbed image over iterations. We select a validation image from Fashion-MNIST. Using the image's two largest principal components, a two-dimensional hyperplane $\subset \mathbb{R}^{28\times28}$ intersects the image with the image at the center.

Vector arrows visualize the generative feedback's movement on the hyperplane's position. In Figure 2.6 (a), we find that generative feedback perturbs samples across decision boundaries toward the validation image. This demonstrates that the CNN-F's generative feedback can restore perturbed images to their uncorrupted objects.

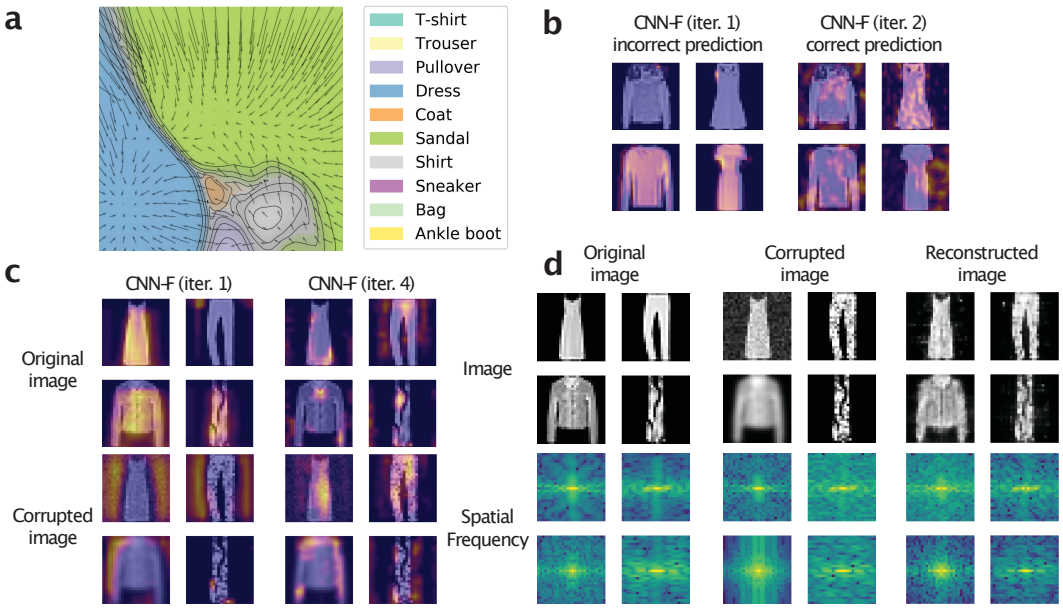


Figure 2.6: **The generative feedback in CNN-F models restores perturbed images.** a) The decision cell cross-sections for a CNN-F trained on Fashion-MNIST. Arrows visualize the feedback direction on the cross-section. b) Fashion-MNIST classification accuracy on PGD adversarial examples; Grad-CAM activations visualize the CNN-F model's attention from incorrect (iter. 1) to correct predictions (iter. 2). c) Grad-CAM activations across different feedback iterations in the CNN-F. d) From left to right: clean images, corrupted images, and images restored by the CNN-F's feedback.

We further explore this principle with regard to adversarial examples. The CNN-F model can correct initially wrong predictions. Figure 2.6 (b) uses Grad-CAM activations to visualize the network's attention from an incorrect prediction to a correct prediction on PGD-40 adversarial samples (Selvaraju et al., 2017). To correct predictions, the CNN-F model does not initially focus on specific features. Rather, it either identifies the entire object or the entire image. With generative feedback, the CNN-F begins to focus on specific features. This is reproduced in clean images as well as images corrupted by blurring and additive noise 2.6 (c). Furthermore, with these perceptible corruptions, the CNN-F model can reconstruct the clean image with generative feedback 2.6 (d). This demonstrates that the generative feedback is one mechanism that restores perturbed images.

Adversarial Training

Adversarial training is a well established method to improve adversarial robustness of a neural network (Madry et al., 2018). Adversarial training often solves a minimax optimization problem where the attacker aims to maximize the loss and the model parameters aims to minimize the loss. In this section, we show that CNN-F can be combined with adversarial training to further improve the adversarial robustness.

Training Methods. Figure 2.7 illustrates the loss design we use for CNN-F adversarial training. Different from standard adversarial training on CNNs, we use cross-entropy loss on both clean images and adversarial images. In addition, we add reconstruction loss between generated features of adversarial samples from iterative feedback and the features of clean images in the first forward pass.

Experimental Setup. We train the CNN-F on Fashion-MNIST and CIFAR-10 datasets respectively. For Fashion-MNIST, we train a network with 4 convolution layers and 3 fully-connected layers. We use 2 convolutional layers to encode the image into feature space and reconstruct to that feature space. For CIFAR-10, we use the WideResNet architecture (Zagoruyko and Komodakis, 2016) with depth 40 and width 2. We reconstruct to the feature space after 5 basic blocks in the first network block. For more detailed hyper-parameter settings, please refer to Appendix A.2. During training, we use PGD-7 to attack the first forward pass of CNN-F to obtain adversarial samples. During testing, we also perform SPSA (Uesato et al., 2018) and transfer attack in addition to PGD attack to prevent the gradient obfuscation (Athalye et al., 2018) issue when evaluating adversarial robustness of a model. In

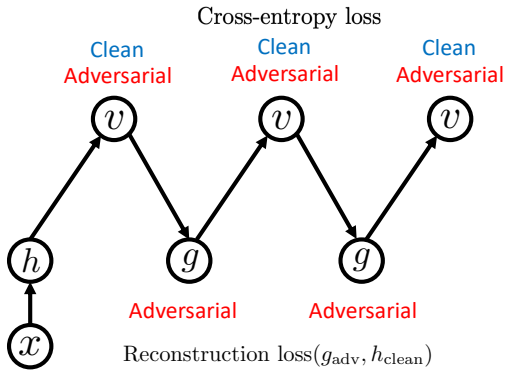


Figure 2.7: **Loss design for CNN-F adversarial training**, where v stands for the logits. x, h and g are input image, encoded feature, and generated feature, respectively.

the transfer attack, we use the adversarial samples of the CNN to attack CNN-F.

Main Results. CNN-F further improves the robustness of CNN when combined with adversarial training. Table 2.2 and Table 2.3 list the adversarial accuracy of CNN-F against several attack methods on Fashion-MNIST and CIFAR-10. On Fashion-MNIST, we train the CNN-F with 1 iterations. On CIFAR-10, we train the CNN-F with 2 iterations. We report two evaluation methods for CNN-F: taking the logits from the last iteration (last), or taking the average of logits from all the iterations (avg). We also report the lowest accuracy among all the attack methods with bold font to highlight the weak spot of each model. In general, we find that the CNN-F tends to be more robust to end-to-end attack compared with attacking the first forward pass. This corresponds to the scenario where the attacker does not have access to internal iterations of the CNN-F. Based on different attack scenarios, we can tune the hyper-paramters and choose whether averaging the logits or outputting the logits from the last iteration to get the best robustness performance (Appendix A.2).

Table 2.2: Adversarial accuracy on Fashion-MNIST over 3 runs. $\epsilon = 0.1$.

	Clean	PGD (first)	PGD (e2e)	SPSA (first)	SPSA (e2e)	Transfer	Min
CNN	$\mathbf{89.97 \pm 0.10}$	77.09 ± 0.19	77.09 ± 0.19	87.33 ± 1.14	87.33 ± 1.14	—	77.09 ± 0.19
CNN-F (last)	89.87 ± 0.14	79.19 ± 0.49	78.34 ± 0.29	87.10 ± 0.10	87.33 ± 0.89	82.76 ± 0.26	78.34 ± 0.29
CNN-F (avg)	89.77 ± 0.08	$\mathbf{79.55 \pm 0.15}$	$\mathbf{79.89 \pm 0.16}$	$\mathbf{88.27 \pm 0.91}$	$\mathbf{88.23 \pm 0.81}$	$\mathbf{83.15 \pm 0.17}$	$\mathbf{79.55 \pm 0.15}$

Table 2.3: Adversarial accuracy on CIFAR-10 over 3 runs. $\epsilon = 8/255$.

	Clean	PGD (first)	PGD (e2e)	SPSA (first)	SPSA (e2e)	Transfer	Min
CNN	79.09 ± 0.11	42.31 ± 0.51	42.31 ± 0.51	66.61 ± 0.09	$\mathbf{66.61 \pm 0.09}$	—	42.31 ± 0.51
CNN-F (last)	78.68 ± 1.33	$\mathbf{48.90 \pm 1.30}$	49.35 ± 2.55	68.75 ± 1.90	51.46 ± 3.22	66.19 ± 1.37	$\mathbf{48.90 \pm 1.30}$
CNN-F (avg)	$\mathbf{80.27 \pm 0.69}$	48.72 ± 0.64	$\mathbf{55.02 \pm 1.91}$	$\mathbf{71.56 \pm 2.03}$	58.83 ± 3.72	$\mathbf{67.09 \pm 0.68}$	48.72 ± 0.64

2.4 Related Works

Robust Neural Networks with Latent Variables. Latent variable models are a unifying theme in robust neural networks. The consciousness prior (Bengio, 2019) postulates that natural representations—such as language—operate in a low-dimensional space, which may restrict expressivity but also may facilitate rapid learning. If adversarial attack introduce examples outside this low-dimensional manifold, latent variable models can map these samples back to the manifold. A related mechanism for robustness is state reification (Lamb et al., 2019). Similar to self-consistency, state reification models the distribution of hidden states over the

training data. It then maps less likely states to more likely states. MagNet and Denoising Feature Matching introduce similar mechanisms: using autoencoders on the input space to detect adversarial examples and restore them in the input space (Meng and Chen, 2017; Warde-Farley and Bengio, 2017). Lastly, Defense-GAN proposes a generative adversarial network to approximate the data manifold (Samangouei et al., 2018). CNN-F generalizes these themes into a Bayesian framework. Intuitively, CNN-F can be viewed as an autoencoder. In contrast to standard autoencoders, CNN-F requires stronger constraints through Bayes rule. CNN-F—through self-consistency—constrains the generated image to satisfy the *maximum a posteriori* on the predicted output.

Computational Models of Human Vision. Recurrent models and Bayesian inference have been two prevalent concepts in computational visual neuroscience. Recently, Kubilius et al. (2018) proposed CORnet as a more accurate model of human vision by modeling recurrent cortical pathways. Like CNN-F, they show CORnet has a larger V4 and IT neural similarity compared to a CNN with similar weights. Linsley et al. (2018) suggests hGRU as another recurrent model of vision. Distinct from other models, hGRU models lateral pathways in the visual cortex to global contextual information. While Bayesian inference is a candidate for visual perception, a Bayesian framework is absent in these models. The recursive cortical network (RCN) proposes a hierarchal conditional random field as a model for visual perception (George et al., 2017). In contrast to neural networks, RCN uses belief propagation for both training and inference. With the representational ability of neural networks, we propose CNN-F to approximate Bayesian inference with recurrent circuits in neural networks.

Feedback Networks. Feedback Network (Zamir et al., 2017) uses convLSTM as building blocks and adds skip connections between different time steps. This architecture enables early prediction and enforces hierarchical structure in the label space. Nayebi et al. (2018) uses architecture search to design local recurrent cells and long range feedback to boost classification accuracy. Wen et al. (2018) designs a bi-directional recurrent neural network by recursively performing bottom up and top down computations. The model achieves more accurate and definitive image classification. In addition to standard image classification, neural networks with feedback have been applied to other settings. Wang et al. (2018) propose a feedback-based propagation approach that improves inference in CNN under partial evidence in the multi-label setting. Piekniewski et al. (2016) apply multi-layer perceptrons

with lateral and feedback connections to visual object tracking.

Combining Top-down and Bottom-up Signals in RNNs. Mittal et al. (2020) proposes combining attention and modularity mechanisms to route bottom-up (feed-forward) and top-down (feedback) signals. They extend the Recurrent Independent Mechanisms (RIMs) (Goyal et al., 2019) framework to a bidirectional structure such that each layer of the hierarchy can send information in both bottom-up direction and top-down direction. Our approach uses approximate Bayesian inference to provide top-down communication, which is more consistent with the Bayesian brain framework and predictive coding.

Inference in Generative Classifiers. Sulam et al. (2019) derives a generative classifier using a sparse prior on the layer-wise representations. The inference is solved by a multi-layer basis pursuit algorithm, which can be implemented via recurrent convolutional neural networks. Nimmagadda and Anandkumar (2015) propose to learn a latent tree model in the last layer for multi-object classification. A tree model allows for one-shot inference in contrast to iterative inference.

Target Propagation. The generative feedback in CNN-F shares a similar form as target propagation, where the targets at each layer are propagated backwards. In addition, difference target propagation uses auto-encoder like losses at intermediate layers to promote network invertibility (Meulemans et al., 2020; Lee et al., 2015). In the CNN-F, the intermediate reconstruction loss between adversarial and clean feature maps during adversarial training promotes the feedback to project perturbed image back to its clean version in all resolution scales.

References

Anish Athalye, Nicholas Carlini, and David A. Wagner (2018). "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples". In: *International Conference on Machine Learning*.

Yoshua Bengio (2019). "The consciousness prior". In: *arXiv:1709.08568*.

S. Dodge and L. Karam (2017). "A Study and Comparison of Human and Deep Learning Recognition Performance under Visual Distortions". In: *International Conference on Computer Communications and Networks*.

Michael Eickenberg, Alexandre Gramfort, Gaël Varoquaux, and Bertrand Thirion (2017). "Seeing it all: Convolutional Network Layers Map the Function of the Human Visual System". In: *NeuroImage*.

Gamaleldin Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein (2018). "Adversarial Examples

that Fool Both Computer Vision and Time-Limited Humans". In: *Advances in Neural Information Processing Systems*.

Daniel J. Felleman and David C. Van Essen (1991). "Distributed Hierarchical Processing in the Primate Cerebral Cortex". In: *Cerebral Cortex*.

Dileep George, Wolfgang Lehrach, Ken Kansky, Miguel Lázaro-Gredilla, Christopher Laan, Bhaskara Marthi, Xinghua Lou, Zhaoshi Meng, Yi Liu, Huayan Wang, Alex Lavin, and D. Scott Phoenix (2017). "A Generative Vision Model that Trains with High Data Efficiency and Breaks Text-based CAPTCHAs". In: *Science*.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy (2015). "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations*.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf (2019). "Recurrent Independent Mechanisms". In: *arXiv:1909.10893*.

Tomoyasu Horikawa and Yukiyasu Kamitani (2017). "Hierarchical Neural Representation of Dreamed Objects Revealed by Brain Decoding with Deep Neural Network Features". In: *Frontiers in Computational Neuroscience*.

Kohitij Kar, Jonas Kubilius, Kailyn Schmidt, Elias B Issa, and James J DiCarlo (2019). "Evidence that Recurrent Circuits are Critical to the Ventral stream's Execution of Core Object Recognition Behavior". In: *Nature Neuroscience*.

Tim C Kietzmann, Courtney J Spoerer, Lynn KA Sörensen, Radoslaw M Cichy, Olaf Hauk, and Nikolaus Kriegeskorte (2019). "Recurrence is Required to Capture the Representational Dynamics of the Human Visual System". In: *the Proceedings of the National Academy of Sciences*.

David C Knill and Whitman Richards (1996). *Perception as Bayesian Inference*. Cambridge University Press.

Peter Kok, Janneke FM Jehee, and Floris P De Lange (2012). "Less is More: Expectation Sharpens Representations in the Primary Visual Cortex". In: *Neuron*.

Jonas Kubilius, Martin Schrimpf, Aran Nayebi, Daniel Bear, Daniel LK Yamins, and James J DiCarlo (2018). "CORnet: Modeling the Neural Mechanisms of Core Object Recognition". In: *bioRxiv preprint: 10.1101/408385*.

Alex Lamb, Jonathan Binas, Anirudh Goyal, Sandeep Subramanian, Ioannis Mitliagkas, Denis Kazakov, Yoshua Bengio, and Michael C. Mozer (2019). "State-Reification Networks: Improving Generalization by Modeling the Distribution of Hidden Representations". In: *International Conference on Machine Learning*.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio (2015). "Difference Target Propagation". In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.

Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre (2018). "Learning Long-range Spatial Dependencies with Horizontal Gated Recurrent Units". In: *Advances in Neural Information Processing Systems*.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu (2018). "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations*.

Dongyu Meng and Hao Chen (2017). "MagNet: a Two-pronged Defense against Adversarial Examples". In: *ACM Conference on Computer and Communications Security*.

Alexander Meulemans, Francesco S Carzaniga, Johan AK Suykens, João Sacramento, and Benjamin F Grewe (2020). "A Theoretical Framework for Target Propagation". In: *arXiv:2006.14331*.

Sarthak Mittal, Alex Lamb, Anirudh Goyal, Vikram Voleti, Murray Shanahan, Guillaume Lajoie, Michael Mozer, and Yoshua Bengio (2020). "Learning to Combine Top-Down and Bottom-Up Signals in Recurrent Neural Networks with Attention over Modules". In: *International Conference on Machine Learning*.

Aran Nayebi, Daniel Bear, Jonas Kubilius, Kohitij Kar, Surya Ganguli, David Sussillo, James J DiCarlo, and Daniel L Yamins (2018). "Task-driven Convolutional Recurrent Models of the Visual System". In: *Advances in Neural Information Processing Systems*.

Tan Nguyen, Nhat Ho, Ankit Patel, Anima Anandkumar, Michael I. Jordan, and Richard G. Baraniuk (2018). "A Bayesian Perspective of Convolutional Neural Networks through a Deconvolutional Generative Model". In: *arXiv:1811.02657*.

Tejaswi Nimmagadda and Anima Anandkumar (2015). "Multi-object Classification and Unsupervised Scene Understanding using Deep Learning Features and Latent Tree Probabilistic Models". In: *arXiv:1505.00308*.

Filip Piekniewski, Patryk Laurent, Csaba Petre, Micah Richert, Dimitry Fisher, and Todd Hylton (2016). "Unsupervised Learning from Continuous Video in a Scalable Predictive Recurrent Network". In: *arXiv:1607.06854*.

Rajesh P. N. Rao and Dana H. Ballard (1999). "Predictive Coding in the Visual Cortex: a Functional Interpretation of Some Extra-classical Receptive-field Effects". In: *Nature Neuroscience*.

Pouya Samangouei, Maya Kabkab, and Rama Chellappa (2018). "Defense-GAN: Protecting Classifiers against Adversarial Attacks using Generative Models". In: *International Conference on Learning Representations*.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra (2017). "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization". In: *International Conference on Computer Vision*.

Jeremias Sulam, Aviad Aberdam, Amir Beck, and Michael Elad (2019). “On Multi-layer Basis Pursuit, Efficient Algorithms and Convolutional Neural Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna (2016). “Rethinking the Inception Architecture for Computer Vision”. In: *IEEE / CVF Computer Vision and Pattern Recognition Conference*.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2014). “Intriguing Properties of Neural Networks”. In: *International Conference on Learning Representations*.

Jonathan Uesato, Brendan O’Donoghue, Aaron van den Oord, and Pushmeet Kohli (2018). “Adversarial Risk and the Dangers of Evaluating against Weak Attacks”. In: *International Conference on Machine Learning*.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky (2016). “Instance Normalization: The Missing Ingredient for Fast Stylization”. In: *arXiv:1607.08022*.

Tianlu Wang, Kota Yamaguchi, and Vicente Ordonez (2018). “Feedback-prop: Convolutional Neural Network Inference under Partial Evidence”. In: *IEEE / CVF Computer Vision and Pattern Recognition Conference*.

David Warde-Farley and Yoshua Bengio (2017). “Improving Generative Adversarial Networks With Denoising Feature Matching”. In: *International Conference on Learning Representations*.

Haiguang Wen, Kuan Han, Junxing Shi, Yizhen Zhang, Eugenio Culurciello, and Zhongming Liu (2018). “Deep Predictive Coding Network for Object Recognition”. In: *International Conference on Machine Learning*.

Sergey Zagoruyko and Nikos Komodakis (2016). “Wide Residual Networks”. In: *arXiv:1605.07146*.

Amir R. Zamir, Te-Lin Wu, Lin Sun, William B. Shen, Bertram E. Shi, Jitendra Malik, and Silvio Savarese (2017). “Feedback Networks”. In: *IEEE / CVF Computer Vision and Pattern Recognition Conference*.

Chapter 3

CERTIFIABLY ROBUST NEURAL NETWORKS WITH EFFICIENT LOCAL LIPSCHITZ BOUNDS

Yujia Huang, Huan Zhang, Yuanyuan Shi, J. Zico Kolter, and Anima Anandkumar (2021). "Training Certifiably Robust Neural Networks with Efficient Local Lipschitz Bounds". In: *Advances in Neural Information Processing Systems* 34, pp. 22745–22757. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/c055dcc749c2632fd4dd806301f05ba6-Paper.pdf.

3.1 Introduction

With the ever-growing deployment of deep neural networks, formal robustness guarantees are needed in many safety-critical applications. Strategies to improve robustness such as adversarial training only provide empirical robustness, without formal guarantees, and many existing adversarial defenses have been successfully broken using stronger attacks (Athalye et al., 2018). In contrast, certified defenses give formal robustness guarantees that any norm-bounded adversary cannot alter the prediction of a given network.

Bounding the global Lipschitz constant of a neural network is a computationally efficient and scalable approach to provide certifiable robustness guarantees (Cissé et al., 2017; Qian and Wegman, 2019; Leino et al., 2021). The global Lipschitz bound is typically computed as the product of the spectral norm of each layer. However, this bound can be quite loose because it needs to hold for *all* points from the input domain, including those inputs that are far away from each other. Training a network while constraining this loose bound often imposes to high a degree of regularization and reduces network capacity. It leads to considerably lower clean accuracy in certified training compared to standard and adversarial training (Huster et al., 2018; Madry et al., 2018).

A local Lipschitz constant, on the other hand, bounds the norm of output perturbation only for inputs from a small region, usually selected as a neighborhood around each data point. It produces a tighter bound by considering the geometry in a local region and often yields much better robustness certification (Hein and Andriushchenko, 2017; Zhang et al., 2019). Unfortunately, computing the exact local

Lipschitz constant is NP-complete (Katz et al., 2017). Obtaining reasonably tight local Lipschitz bounds via semidefinite programming (Fazlyab et al., 2019) or mixed integer programming (Jordan and Dimakis, 2020) is typically only applicable to small, *previously trained networks* since it is difficult to parallelize the optimization solver and make it differentiable for training. On the other hand, many existing certified defense methods have achieved success by using a training-based approach with a relatively weak but efficient bound (Gowal et al., 2018; Mirman et al., 2018; Zhang et al., 2020). Therefore, to incorporate local Lipschitz bound in training, a computationally efficient and training-friendly method must be developed.

Our contributions. We propose an efficient method to incorporate a local Lipschitz bound in training deep networks, by considering the interactions between an activation layer such as a Rectified Linear Unit (ReLU) layer and a linear (or convolution) layer. Our bound is calculated per data point, leading to activation function outputs that are either constant or vary with input perturbations. If the outputs of some activation neurons are constant under local perturbation, we eliminate the corresponding rows in the previous weight layer and the corresponding columns in the next weight layer, and then compute the spectral norm of the reduced matrix.

Our main insight is to use training to make the proposed local Lipschitz bound tight. This is different from existing works that find local Lipschitz bound for a fixed network (Zhang et al., 2019; Fazlyab et al., 2019; Jordan and Dimakis, 2020). Instead, we aim to enable a network to learn to tighten our proposed local Lipschitz bound during training. To achieve this, we propose to clip the activation function with an individually learnable threshold θ. Take ReLU for example, the output of a "clipped" ReLU becomes a constant when input is greater than this threshold (see Figure 3.1). Once the input of the ReLU is greater than the threshold or less than 0, then this ReLU neuron does not contribute to the local Lipschitz constant, and thus the corresponding row or column of weight matrices can be removed. We also apply this method to non-ReLU activation functions such as MaxMin (Anil et al., 2019) to create constant output regions. Additionally, we also use a hinge loss function to encourage more neurons to have constant outputs. Our method can be used as a plug-in module in existing certifiable training algorithms that involve computing Lipschitz bound. Our contributions can be summarized as:

- To the best of our knowledge, we are the first to incorporate a local Lipschitz bound during training for certified robustness. Our bound is provably tighter than the global Lipschitz bound and is also computationally efficient for training.

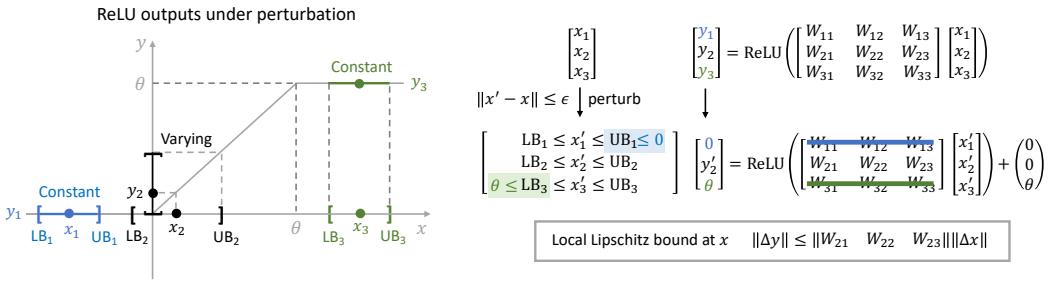


Figure 3.1: **Illustration of tighter (local) Lipschitz constant with bounded ReLU.**

- We propose to use activation functions with learnable threshold to encourage more fixed neurons during training, which assists the network to learn to tighten our bound. We show that more than 45% rows and columns can be removed from weight matrices.

- We consistently outperform state-of-the-art Lipschitz based certified defense methods for ℓ_2 norm robustness. On CIFAR-10 with perturbation $\epsilon = \frac{36}{255}$, we obtain 54.3% verified accuracy with ReLU activation function and 60.7% accuracy with MaxMin (Anil et al., 2019), outperforming the SOTA baselines, and also achieve better clean accuracy. Our code is available at `https://github.com/yjhuangcd/local-lipschitz`.

3.2 Related Works

Bounds on Local Lipschitz Constant. A *sound upper bound* of local Lipschitz constant (simply referred to as "local Lispschitz bound" in our paper) is a crucial property to determine the robustness of a classifier. Finding an exact local Lipschitz constant for a neural network is generally NP hard (Virmaux and Scaman, 2018), so most works focus on finding a sound upper bound. Hein and Andriushchenko (2017) derived an analytical bound for 2 layer neural networks and found that local Lipschitz bounds could be much tighter than the global one and give better robustness certificates. RecurJac (Zhang et al., 2019) is a recursive algorithm that analyzes the local Lipschitz constant in a neural network using a bound propagation (Zhang et al., 2020) based approach. FastLip (Weng et al., 2018) is a special and weaker form of RecurJac. Fazlyab et al. (2019) used a stronger semidefinite relaxation to compute a tighter bound of local Lipschitz constant. Jordan and Dimakis (2020) formulated the computation of Lipschitz as an mixed integer linear programming (MILP) problem and they were able to solve the exact local Lipschitz constant. Although these approaches can obtain reasonably tight and sound local Lipschitz constants, none

of them have been demonstrated effective for training a certifiably robust network, where high efficiency and scalability are required. Note that although an empirical estimate of local Lipschitz constant can be easily found via gradient ascent (e.g., the local lower bounds reported in (Leino et al., 2021)), it is not a sound bound and does not provide certifiable robustness guarantees.

Certifiably Robust Training using Lipschitz Constants. The Lipschitz constant plays a central role in many works on training a certifiably robust neural network, especially for ℓ_2 norm robustness. Since naturally trained networks usually have very large global Lipschitz constant bounds (Szegedy et al., 2014), most existing works train the network to encourage small a Lipschitz bound. Cissé et al. (2017) designed networks with orthogonal weights, whose Lipschitz constants are exactly 1. As this can be too restrictive, later works mostly use power iteration to obtain per-layer induced norms, whose product is a Lipschitz constant. Lipschitz Margin Training (LMT) (Tsuzuku et al., 2018) and Globally-Robust Neural Networks (Gloro) (Leino et al., 2021) both upper bound the worst margin via global Lipschitz constant with different loss functions. LMT constructs a new logit by adding the worst margin to all its entries except the ground truth class. Gloro construct a new logit with one more class than the original logit vector, determines whether the input sample can be certified. However, these approaches did not exploit the available local information to tighten Lipschitz bound and improve certified robustness. Box constrained propagation (BCP) (Lee et al., 2020) achieves a tighter outer bound than global Lipschitz based outer bound, by taking local information into consideration via interval bound (box) propagation. They compute the worst case logit based on the intersection of a (global) ball and a (local) box. Although box propagation considers local information, the ball propagation still uses global Lipschitz constant, and its improvement is still limited with low clean accuracy.

Other Certified Defenses. Besides using Lipschitz constants, one of the most popular certifiable defense against ℓ_∞ norm bounded inputs is via the convex outer adversarial polytope (Wong and Kolter, 2018; Wong et al., 2018). Mirman et al. (2018) takes a similar approach via abstract interpretation. These methods uses linear relaxations of neural networks to compute an outer bound at the final layer. However, because the convex relaxations employed are relatively expensive, these methods are typically slow to train. A simple and fast certifiable defense for ℓ_∞ norm bounded inputs is interval bound propagation (IBP) (Gowal et al., 2018; Mirman et al., 2018). Since the IBP bound can be quite loose for general networks, its

good performance relies on appropriate hyper-parameters. CROWN-IBP (Zhang et al., 2020) outperforms previous methods by combining IBP bound in a forward bounding pass and a tighter linear relaxation bound in a backward bound pass. Shi et al. (2021) improved IBP with better initialization to accelerate training. Additionally, randomized smoothing (Cohen et al., 2019; Li et al., 2019; Lecuyer et al., 2019) is a probabilistic method to certify ℓ_2 norm robustness with arbitrarily high confidence. The prediction of a randomized smooth classifier is the most likely prediction returned by the base classifier that is fed by samples from a Gaussian distribution. Salman et al. (2019) further improves the performance of randomized smoothing via adversarial training.

3.3 Efficient Local Lipschitz Bound

We begin with notations and background for Lipschitz bound. We introduce our method for local Lipschitz bound computation in Section 3.3. Then we introduce how to incorporate our efficient local Lipschitz bound in robust training (Section 3.3).

Notation and Background

Notations. We denote the Euclidean norm of a vector x as $\|x\|$ and $\|A\|$ is the spectral norm of matrix A. Subscript of vector x denotes element, i.e., x_i is the i-th element of x. We use LB^l and UB^l to denote lower bounds and upper bounds of pre-ReLU activation values for layer l.

Definition 3.3.1. The Lipschitz constant of a function $f : \mathbb{R}^d \to \mathbb{R}^m$ over an open set $\mathcal{X}$ is defined as,

$$L(f, \mathcal{X}) := \sup_{x,y \in \mathcal{X}, x \neq y} \frac{||f(y) - f(x)||}{||x - y||}.$$

If $L(f, \mathcal{X})$ exists and is finite, we say that f is Lipschitz continuous over $\mathcal{X}$. Suppose $\mathcal{X} = \text{dom}(f)$, $L(f, \mathcal{X})$ is the **global** Lipschitz constant of f; if $\mathcal{X}$ is defined as the ϵ-ball at point x, i.e., $\mathcal{X} := \{x' | ||x - x'|| \leq \epsilon\}$, then $L(f, \mathcal{X})$ is the **local** Lipschitz constant of f at x.

Global Lipschitz bound in existing works. Consider a L-layer ReLU neural network which maps input x to output $z^{L+1} = F(x; W)$ using the following architecture, for $l = 1, ..., L - 1$

$$z^1 = x; \quad z^{l+1} = \phi(W^l z^l); \quad z^{L+1} = W^L z^L \tag{3.1}$$

where $W = \{W^{1:L}\}$ are the parameters, and $\phi(\cdot) = \max(\cdot, 0)$ is the element-wise ReLU activation functions. Here we consider the bias parameters to be zero because they do not contribute to the Lipschitz bound. Since the Lipschitz constant of ReLU activation $\phi(\cdot)$ is equal to 1, a global Lipschitz bound of F is,

$$L_{\text{glob}} \leq ||W^L|| \cdot ||W^{L-1}|| \cdots ||W^1|| \tag{3.2}$$

where $||W^l||$ equals the spectral norm (maximum singular value) of the weight matrix W^l. However, the global Lipchitz bound ignores the highly nonlinear property of deep neural networks.

In what follows, we introduce our method that considers the interaction between ReLU and linear layer to obtain a tighter local Lipchitz bound in a computationally efficient way, that allows us to train a certifiably robust network using local Lipschitz bounds.

Our Approach for Efficient Local Lipschitz Bound

In this section, we use ReLU as an example to describe how we compute our efficient local Lipschitz bound. We will discuss how to apply our method on other types of activation functions in Section 3.3. To exploit the piece-wise linear properties of ReLU neurons, we discuss the outputs of ReLU case by case. Intuitively, if the input of a ReLU neuron is always less or equal to zero, its output will always be zero, which is a constant and not contributing to Lipschitz bound. If the input of a ReLU's can sometimes be greater than zero, the ReLU output will vary based on the input.

We define diagonal indicator matrices $I^l_{\text{V}}(z^l)$ to represent the entries where the ReLU outputs are *varying* and $I^l_{\text{C}}(z^l)$ for entries where the ReLU outputs are *constant* under perturbation. Here $z^l \in \mathbb{R}^{d_l}$ denotes the feature map of input x at layer l. Throughout this paper, unless otherwise mentioned, the indicator matrix is a function of the feature value z^l, evaluated at a given input x.

Given an input perturbation $\|x' - x\| \leq \epsilon$, suppose $z^l(x')$ is bounded element-wise as $\text{LB}^l \leq z^l(x') \leq \text{UB}^l$, we define diagonal matrix I^l_{V} and I^l_{C} as:

$$I^l_{\text{V}}(i,i) = \begin{cases} 1 & \text{if } \text{UB}^l_i > 0 \\ 0 & \text{otherwise} \end{cases}, \quad I^l_{\text{C}}(i,i) = \begin{cases} 1 & \text{if } \text{UB}^l_i \leq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{3.3}$$

By this definition, the ReLU output can either be constant or vary with respect to input perturbation. Hence we have $I^l_{\text{V}} + I^l_{\text{C}} = I$, where I is the identity matrix. LB

and UB can be obtained cheaply from interval bound propagation (Gowal et al., 2018) or other bound propagation mechanisms (Wong and Kolter, 2018; Zhang et al., 2018).

A crucial observation is that to compute the local Lipschitz bound, we only need to consider the ReLU neurons which are non-fixed. The fixed ReLU neurons are always zero (locally, in the prescribed neighborhood around x) and thus have no impact to final outcome. We define a diagonal matrix D_{V} to represent the ReLU outputs that are varying. Then, a neural network function (denoted as $F(x; W)$) can be rewritten as:

$$\begin{aligned} F(x; W) &= W^L D_{\mathrm{V}}^{L-1} W^{L-1} \cdots D_{\mathrm{V}}^1 W^1 x \\ &= (W^L I_{\mathrm{V}}^{L-1}) D_{\mathrm{V}}^{L-1} (I_{\mathrm{V}}^{L-1} W^{L-1} I_{\mathrm{V}}^{L-2}) \cdots D_{\mathrm{V}}^1 (I_{\mathrm{V}}^1 W^1) x \end{aligned} \tag{3.4}$$

where

$$D_{\mathrm{V}}^l(i,i) = \begin{cases} \mathbb{1}(\mathrm{ReLU}(z_i^l) > 0) & \text{if } I_{\mathrm{V}}^l(i,i) = 1 \\ 0 & \text{if } I_{\mathrm{V}}^l(i,i) = 0 \end{cases}, \tag{3.5}$$

where $\mathbb{1}$ denotes an indicator function.

Note here that we ignore bias terms for simplicity. Based on (3.4), an important insight used in our approach is that by combining the ReLU function with weight matrix, we have the opportunity to tighten Lipschitz bound by considering $I_{\mathrm{V}}^l W^l I_{\mathrm{V}}^{l-1}$ as a whole based on whether there are ReLU outputs stay at constant under perturbation. Importantly, since D_{V}^l depends on UB, (3.4) only holds in a local region of x, which leads to a local Lipschitz constant bound at input x:

$$L_{\text{local}}(x) \leq \|W^L I_{\mathrm{V}}^{L-1}\| \|I_{\mathrm{V}}^{L-1} W^{L-1} I_{\mathrm{V}}^{L-2}\| \cdots \|I_{\mathrm{V}}^1 W^1\|. \tag{3.6}$$

The following theorem states that the local Lipchitz bound calculated via (3.6) is always tighter than the global Lipchitz bound in Eq (3.2), for all inputs.

Theorem 3.3.1 (Tighter Lipchitz Bound)**.** *For any input $x \in \mathbb{R}^n$ and L-layer ReLU neural network $F(x; W)$, the local Lipchitz bound calculated via* (3.6) *in any neighborhood of x is no larger than the global Lipchitz bound in Eq* (3.2)*, i.e., $\forall x, L_{local}(x) \leq L_{glob}$.*

The proof of Theorem 3.3.1 leverages the following proposition.

Proposition 3.3.1. *If a column and/or row is added to a matrix, then the matrix spectral norm (maximum singular value) will be no less than the spectral norm of the original matrix. That is, given matrix $A \in \mathbb{R}^{m\times n}$, and $y \in R^m, z \in R^n$, then*

$$\sigma_{max}([A|y]) \geq \sigma_{max}(A)\,, \sigma_{max}(\begin{bmatrix} A \\ z \end{bmatrix}) \geq \sigma_{max}(A). \tag{3.7}$$

Proof of Proposition 3.3.1. Let $A' = \begin{bmatrix} A \\ z \end{bmatrix}$. The singular value of A' is defined as the square roots of the eigenvalues of $A'^T A'$, where $A'^T A' = A^T A + z^T z \geq A^T A$, that simply imply $||A'|| = \sigma_{max}(A') \geq \sigma_{max}(A) = ||A||$. Similar holds for $A' = [A|y]$. □

By Proposition 3.3.1, it is straightforward to show that $\|I_{\mathrm{V}}^{L-1} W^{L-1} I_{\mathrm{V}}^{L-2}\| \leq \|W^{L-1}\|$ since the left hand side is the spectral norm of the reduced matrix, after removing corresponding rows/columns in W^{L-1} where the neuron output under local perturbation is constant. Therefore, the product of spectral norm of the reduced matrices is no larger than the product of the spectral norm of raw weight matrices, which leads to $\forall x, L_{local}(x) \leq L_{glob}$.

Training for Tight Local Lipschitz

To encourage the network to learn which rows and columns need to be eliminated to make local Lipschitz bound tighter, we combine our local Lipschitz bound computation with certifiably robust training. This is different from existing works leveraging optimization tools to find local Lipschitz bound for a fixed network (Fazlyab et al., 2019; Jordan and Dimakis, 2020). By training with the proposed local Lipschitz bound, we can achieve good certified robustness on large neural networks.

More precisely, using our local Lipschitz bound, we can obtain the worst case logit z^* that is used to form a robust loss for training: $\mathbb{E}_{(x,y)\sim\mathcal{D}}\mathcal{L}(z^*(x), y)$, where $\mathcal{L}$ is the cross entropy loss function, (x, y) is the image and label pair from the training datasets. A simple way to compute the worst logit is $z_i^* = z_i + \sqrt{2}\epsilon L_{\text{Local}}$ for $i \neq y$, $z_y^* = z_y$ (see (Tsuzuku et al., 2018)). Our approach is also compatible with tighter bounds on the worst case logit, such as the one used in BCP (Lee et al., 2020). To give the network more capability to learn to tighten our proposed local Lipschitz bound, we propose the following approaches:

Allowing More Eliminated Rows via ReLUθ . The key to tighten our local Lipschitz bound is to delete rows and columns in weight matrices that align with

singular vectors corresponding to the largest singular value. To encourage more rows and columns to be deleted, we need to have more ReLU outputs to be at constant under perturbation. Standard ReLU is only lower bounded by zero, but is not upper bounded. If we can set an "upper bound" of ReLU output, we can have more neurons have fixed outputs at this upper bound. An upper bounded ReLU unit called ReLU6 is proposed in (Howard et al., 2017), where the maximum output is set to a constant 6. Different from ReLU6 that sets a constant maximum output threshold, we make the threshold to be a learnable parameter. We name the new type of activation function ReLUθ , which is defined as:

$$\text{ReLU}\theta\ (z_i;\theta_i) = \begin{cases} 0, & \text{if } z_i <= 0 \\ z_i, & \text{if } 0 < z_i < \theta_i \\ \theta_i, & \text{if } z_i >= \theta_i \end{cases} \tag{3.8}$$

where θ_i is a learnable upper bound of the ReLU output.

Similar to (3.3), the indicator matrices for the varying outputs of ReLUθ are

$$I_{\text{V}}^l(i,i) = \begin{cases} 1 & \text{if } \text{UB}_i^l > 0 \text{ and } \text{LB}_i^l < \theta_i \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

Depending on the ReLUθ activation status, the output of a ReLUθ neural network is,

$$F(x;W,\theta) = W^L(D_{\text{V}}^{L-1}W^{L-1}\cdots(D_{\text{V}}^1W^1x + D_\theta^1)\cdots + D_\theta^{L-1})\,, \tag{3.10}$$

where D_θ^l denotes the ReLU output fixed at the maximum output value,

$$D_\theta^l(i,i) = \begin{cases} \theta_i & \text{if } I_\theta^l(i,i) = 1 \\ 0 & \text{if } I_\theta^l(i,i) = 0 \end{cases}, \quad I_\theta^l(i,i) = \begin{cases} 1 & \text{if } \text{LB}_i^l \geq \theta_i \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

The local Lipchitz bound is still calculated as (3.6). However, the bound can be potentially learned tighter because we encourage ReLU outputs to be constant in both directions, and there could be less varying outputs in Eq (3.9) than in Eq (3.3).

Extension to Non-ReLU Activation Functions. Our local Lipschitz bound can be applied on non-ReLU activation functions. The key is to create constant output regions for the activation function and delete the corresponding rows or columns in the weight matrices. Since the MaxMin activation function (Anil et al., 2019) has been shown to outperform ReLU on certified robustness (Leino et al., 2021;

Trockman and Kolter, 2021), we take MaxMin as an example to explain how to apply our local Lipschitz bound. Let x_1 and x_2 be two groups of the input, the output of MaxMin is $\max(x_1, x_2), \min(x_1, x_2)$. To exploit local Lipschitz, we created a clipped version of MaxMin, similar to ReLUθ. The output of the clipped MaxMin is $\min(\max(x_1, x_2), a), \max(\min(x_1, x_2), b)$, where a is a learnable upper threshold for the max output in MaxMin and b is a learnable lower threshold for the min output in MaxMin. Box propagation rule through MaxMin is straightforward to derive, so we can get the box bound on each entry after MaxMin. If the lower bounds of the Max entries are bigger than the upper threshold a, or the upper bounds of the Min entries are smaller than the lower threshold b, we can delete the corresponding columns in the successive matrix (similar to the procedure for ReLU networks).

Encouraging Fixed Neurons via a Sparsity Loss. To encourage more rows and columns to be deleted in the weight matrices, we design a sparsity loss to regularize the neural network towards this goal. For a ReLU neural network, assuming that the i-th entry of the feature map at layer l is bounded by $\text{LB}_i^l \leq z_i^l \leq \text{UB}_i^l$, we hope the neural network can learn to make as many UB_i to be smaller than zero and LB_i to be larger than θ_i without sacrificing too much of classification accuracy. For a MaxMin neural network, let LB_{max} be the lower bounds of the Max entries, and UB_{min} be the upper bounds of the Min entries. We hope the neural network can learn to make as many LB_{max} to be larger than the upper threshold a and UB_{min} to be smaller than the lower threshold b. The sparsity losses for ReLU and MaxMin networks are as follows:

$$\mathcal{L}_{\text{sparsity}}^{\text{ReLU}} = \max(0, \text{UB}_i^l) + \max(0, \theta_i - \text{LB}_i^l) \tag{3.12}$$

$$\mathcal{L}_{\text{sparsity}}^{\text{MaxMin}} = \max(0, \text{UB}_{\text{min}} - b) + \max(0, a - \text{LB}_{\text{max}})\,. \tag{3.13}$$

Finally, our full training procedure is presented in Algorithm 2.

Computational Cost. To obtain the local Lipschitz bound, we must perform power iterations to compute the spectral norm of reduced weight matrices for every input and its feature maps at each layer. Compared to other methods that use optimization tools (e.g., SDP, MILP) to bound local Lipschitz, our method is computationally efficient since only matrix vector multiplication is used.

During training, compared to methods that only use global Lipschitz bound, the local Lipschitz bound varies based on the inputs. For global Lipschitz bound, a common practice is to keep track of the iterate vector u in power iteration, and use it to initialize the power iteration in the next training batch. With this initialization,

Algorithm 2 Local Lipschitz based Certifiably Robust Training

Require: Training data $(x, y) \sim \mathcal{D}$, perturbation size ϵ, number of iterations for power method n, a neural network with L layers.

repeat
  Compute the box outer bound $[\text{LB}^l, \text{UB}^l]$ for layers 1 to L.
  Compute indicator matrix I_{V} using Eq (3.9).
  ▷ *Compute local Lipcthiz bound $L(x)$ for every input x using Eq* (3.6)
  for layer from 1 to L **do**
    ▷ *Power method*
    Initialize u^l with the updated u^l from the previous training episode.
    for $i < n$ **do**
      if layer is conv **then**
        $v \leftarrow I^l_{\text{V}}\text{conv}(W^l, I^{l-1}_{\text{V}}u^l)/\|I^l_{\text{V}}\text{conv}(W^l, I^{l-1}_{\text{V}}u^l)\|$.
        $u \leftarrow I^{l-1}_{\text{V}}\text{conv}^\intercal(W^l, I^l_{\text{V}}v)/\|I^{l-1}_{\text{V}}\text{conv}^\intercal(W^l, I^l_{\text{V}}v)\|$.
      else if layer is linear **then**
        $v \leftarrow I^l_{\text{V}}W^l I^{l-1}_{\text{V}}u^l/\|I^l_{\text{V}}W^l I^{l-1}_{\text{V}}u^l\|$.
        $u \leftarrow I^{l-1}_{\text{V}}W^{l\intercal} I^l_{\text{V}}v/\|I^{l-1}_{\text{V}}W^{l\intercal} I^l_{\text{V}}v\|$.
      end if
    end for
    if layer is conv **then**
      $\sigma^l(x) \leftarrow v^l I^l_{\text{V}}\text{conv}(W^l, I^{l-1}_{\text{V}}u^l)$.
    else if layer is linear **then**
      $\sigma^l(x) \leftarrow v^l I^l_{\text{V}}W^l I^{l-1}_{\text{V}}u^l$.
    end if
  end for
  Compute the worst logits using our local Lipschitz bound.
  Update model parameters based on some loss functions (e.g., Cross-entropy loss).
until training ends

Ensure: Parameters of a robust neural network

only a few number of iterations is performed during training (typically the number of iterations is between 1 to 10 (Lee et al., 2020; Leino et al., 2021)). To extend the initialization strategy for u to compute local Lipschitz, we need to keep track of the iterate vectors based on every input and its feature maps for every layer. Fortunately, this vector only provides an initialization for power iteration so it does not need to be stored accurately, and can be stored using low precision tensors. Further extensions could use dimension reduction or compression methods to store these vectors, or learn a network along the way to predict a good initializer for power iteration. An alternative approach is to random initialize u in every mini-batch, but we find that more power iterations need to be performed during training for this approach to have

comparable performance as the using saved u.

During evaluation, to minimize the extra computational cost, we can avoid bound computation for two types of inputs: inputs that can already be certified using global Lipschitz bound or can be attacked by adversaries (e.g., a 100-step PGD attack). In practice, this typically rules out more than 80% samples on CIFAR-10 or larger datasets, and greatly reduce computational cost required to compute local Lipschitz constants. In Section 3.4 we will show more empirical results on this aspect.

3.4 Experiment

In this section, we first show that our method achieves tighter Lipschitz bounds in neural networks. When combined with certifiable training algorithms such as BCP (Lee et al., 2020) and Gloro (Leino et al., 2021), as well as training algorithms using orthogonal convolution and MaxMin activation function (Trockman and Kolter, 2021), our method achieves both higher clean and certified accuracy.

Experiment Setup. We train with our method to certify robustness within a ℓ_2 ball of radius 1.58 on MNIST (LeCun et al., 2010) and $36/255$ on CIFAR-10 (Krizhevsky, Hinton, et al., 2009) and Tiny-Imagenet [1] on various network architectures. We denote neural network architecture by the number of convolutional layers and the number of fully-connected layers. For instance, 6C2F indicates that there are 6 convolution layers and 2 fully-connected layers in the neural network. Networks have ReLU activation function unless mentioned otherwise. For more details and hyper-parameters in training, please refer to Appendix B.3.

Tighter Lipschitz Bound. We compared the training process of our method and BCP (Lee et al., 2020) in Figure 3.2 (a-c). During training, our method uses local Lipschitz bound while BCP uses global Lipschitz bound for robust loss. We also tracked the global Lipschitz bound during our training and the average local Lipschitz bound (computed by our method) during BCP training for comparison. We can see from Figure 3.2 (a) that our local Lipschitz bound is always tighter the global Lipschitz bound. Furthermore, it is crucial to incorporate our bound in training to enable the network to learn to tighten the bound. We can see that if we directly apply our method to a BCP trained network after training, the local Lipschitz bound has much less improvement over global Lipschitz bound. In addition, a tighter local Lipschitz bound by our method allows the neural networks to have larger global Lipschitz bound in the beginning of training. This potentially provides larger model

[1] `https://tiny-imagenet.herokuapp.com`

capacity and eases the training in the early stage. As a consequence, we see a large improvement of both clean loss (Figure 3.2 (b)) and also robust loss (Figure 3.2 (c)) throughout the training.

Sparsity of Varying ReLU Outputs. We examine our 6C2F model trained on CIFAR-10 and report the proportion of varying (non-constant) ReLU outputs at all layers except the last fully-connected layer (Figure 3.3). We compared the proportion with that of a standard CNN (trained with only cross entropy loss on unperturbed inputs) and a robust CNN trained by BCP. As we can see, the standard neural network has the most varying ReLU neurons, indicating that dense varying ReLU outputs may provide larger model capacity for clean accuracy but reduce robustness. Our method has *more* varying ReLU outputs than BCP, while achieving *tighter* local Lipschitz bound than BCP. This indicates that our training method encourages the neural network to learn to delete rows and columns that contribute most to local Lipschitz constant during training, while keeping ReLUs for other rows or columns varying to obtain better natural accuracy.

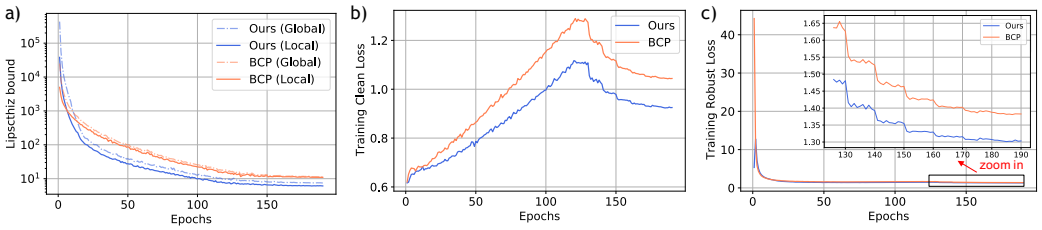


Figure 3.2: **Certifiable training with our method and BCP on CIFAR-10.** a) Global and average local Lipschitz bound during training. Cross entropy loss b) on natural and c) on the worst logits.

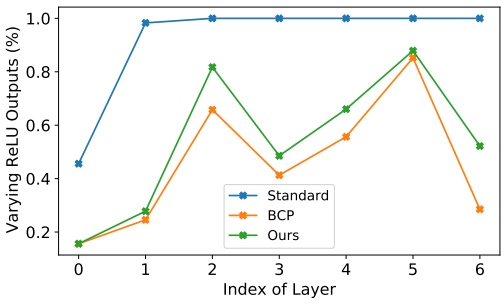


Figure 3.3: **Proportion of ReLU neurons that vary (ReLU outputs are not constants, see definition in Section 3.3) under perturbation.**

Certified Robustness. Our method can be used as a plug-in module in certifiable training algorithms that involves using ℓ_2 Lipschitz constant such as BCP (Lee

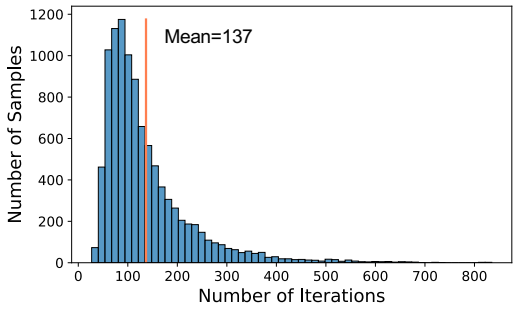


Figure 3.4: **Histogram for the number of power iterations to ensure convergence for the second last linear layer of the 6C2F CIFAR-10 network.**

et al., 2020) and Gloro (Leino et al., 2021). We use Local-Lip-G and Local-Lip-B to denote our method trained with the Gloro loss and BCP loss, respectively (detailed formulation for each loss can be found in Appendix B.2). We compare the performance of our method against competitive baselines on both ReLU neural networks (Lee et al., 2020; Leino et al., 2021; Tsuzuku et al., 2018; Gowal et al., 2018; Wong et al., 2018; Xiao et al., 2019) and MaxMin neural networks (Leino et al., 2021; Trockman and Kolter, 2021). For each method, we report the clean accuracy (accuracy on non-perturbed inputs), the PGD accuracy (accuracy on adversarial inputs generated by PGD attack (Madry et al., 2018)), and the certified accuracy (the proportion of inputs that can be correctly classified and certified within ϵ-ball). For PGD attack, we use 100 steps with step size of $\epsilon/4$. In our experiments, we use box propagation (as done in BCP) to obtain the lower bound and upper bound of every neuron. With our tighter Lipchitz bound, we further improve clean, PGD, and certified accuracy upon BCP and Gloro, and achieve the state-of-the-art performance on certified robustness (Table 3.2). On CIFAR-10 with ReLU activation function, we improved certified accuracy from 51.3% (SOTA) to 54.3%. When MaxMin activation function is used, our local Lipschitz training approach also consistently improves clean, PGD and verified accuracy over baselines on both CIFAR-10 and TinyImageNet datasets.

To demonstrate the effectiveness of incorporating our bound in training, we also use our local bound to directly compute certified accuracy for *pretrained* models using BCP. Our bound improves the certified accuracy of a pretrained BCP model from 51.3% (reported in Table 3.2) to 51.8%. The improvement is less than training with our bound (54.3% in Table 3.2). Therefore, it is crucial to incorporate our bound in training to gain non-trivial robustness improvements.

Initialization Strategy for Power Method. We used two initialization strategies for singular vectors u in power method during training. One option is to store u for all the inputs and feature maps and initialize u in the current training epoch with u from the previous epoch, which requires storage. The other option is to random initialize u. Table 3.1 shows the performance of these two approaches on CIFAR-10 with the 6C2F architecture. The number of power iterations during training is listed in the bracket. Although random initialization is memory-efficient, it needs more power iterations during training to achieve comparable performance compared to the approach of storing u. Too few iterations tend to cause inaccurate singular value and overfitting, resulting in lower certified accuracy.

Table 3.1: Influence of initialization strategy used in power method. All numbers represent the accuracy of the 6C2F architecture on CIFAR-10.

Method	Clean (%)	PGD (%)	Certified (%)
Random init. (2 iters)	76.7	69.0	0.5
Random init. (5 iters)	73.7	66.8	46.0
Random init. (10 iters)	72.0	65.8	51.6
Using saved u (2 iters)	70.7	64.8	**54.3**

Computational Cost during Evaluation Time. Since local Lipschitz bound needs to be evaluated for every input and global Lipschitz bound does not depend on the input, our method involves additional computation cost during certification. Let $u(t)$ be the singular vector computed by power iteration at iteration t, we stop power iteration when $||u(t+1) - u(t)|| \leq 1\text{e}{-3}$. To analyze the computational cost during evaluation time, we plot the histogram of number of iterations for power method to converge for the second last layer in the 6C2F model in Figure 3.4. The average number of iterations for convergence is 137. The histograms for other layers are in Appendix B.3. To reduce the computational cost, we only need to compute local Lipschitz bound for samples that cannot be certified by global Lipschitz bound or cannot be attacked by adversaries. The proportion of those samples is $(100\% - \text{PGD Err} - \text{Global Certified Acc})$. For the 6C2F model on CIFAR-10, the proportion of samples that can be certified using global Lipschitz bound is 51.0%, and the error under PGD attack is 35.2%. Hence we only need to evaluate local Lipschitz bounds on the remaining 13.8% samples, which greatly reduces the overhead of computing the local Lipschitz bounds.

References

Cem Anil, James Lucas, and Roger B. Grosse (2019). "Sorting Out Lipschitz Function Approximation". In: *International Conference on Machine Learning*.

Anish Athalye, Nicholas Carlini, and David A. Wagner (2018). "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples". In: *International Conference on Machine Learning*.

Moustapha Cissé, Piotr Bojanowski, Edouard Grave, Yann N. Dauphin, and Nicolas Usunier (2017). "Parseval Networks: Improving Robustness to Adversarial Examples". In: *International Conference on Machine Learning*.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter (2019). "Certified Adversarial Robustness via Randomized Smoothing". In: *International Conference on Machine Learning*.

Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas (2019). "Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks". In: *Advances in Neural Information Processing Systems*.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli (2018). "On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models". In: *ArXiv preprint*.

Matthias Hein and Maksym Andriushchenko (2017). "Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation". In: *Advances in Neural Information Processing Systems*.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam (2017). "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *ArXiv preprint*.

Todd Huster, Cho-Yu Jason Chiang, and Ritu Chadha (2018). "Limitations of the Lipschitz Constant as a Defense against Adversarial Examples". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer.

Matt Jordan and Alexandros G. Dimakis (2020). "Exactly Computing the Local Lipschitz Constant of ReLU Networks". In: *Advances in Neural Information Processing Systems*.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer (2017). "Reluplex: An efficient SMT Solver for Verifying Deep Neural Networks". In: *International Conference on Computer Aided Verification (CAV)*.

Alex Krizhevsky, Geoffrey Hinton, et al. (2009). *Learning Multiple Layers of Features from Tiny Images*.

Yann LeCun, Corinna Cortes, and C.J. Burges (2010). *MNIST Handwritten Digit Database*.

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana (2019). "Certified Robustness to Adversarial Examples with Differential Privacy". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE.

Sungyoon Lee, Jaewook Lee, and Saerom Park (2020). "Lipschitz-Certifiable Training with a Tight Outer Bound". In: *Advances in Neural Information Processing Systems*.

Klas Leino, Zifan Wang, and Matt Fredrikson (2021). "Globally-Robust Neural Networks". In: *International Conference on Machine Learning*.

Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin (2019). "Certified Adversarial Robustness with Additive Noise". In: *Advances in Neural Information Processing Systems*.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu (2018). "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations*.

Matthew Mirman, Timon Gehr, and Martin T. Vechev (2018). "Differentiable Abstract Interpretation for Provably Robust Neural Networks". In: *International Conference on Machine Learning*.

Haifeng Qian and Mark N. Wegman (2019). "L2-Nonexpansive Neural Networks". In: *International Conference on Learning Representations*.

Hadi Salman, Jerry Li, Ilya P. Razenshteyn, Pengchuan Zhang, Huan Zhang, Sébastien Bubeck, and Greg Yang (2019). "Provably Robust Deep Learning via Adversarially Trained Smoothed Classifiers". In: *Advances in Neural Information Processing Systems*.

Zhouxing Shi, Yihan Wang, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh (2021). "Fast Certified Robust Training via Better Initialization and Shorter Warmup". In: *Advances in Neural Information Processing Systems*.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2014). "Intriguing Properties of Neural Networks". In: *International Conference on Learning Representations*.

Asher Trockman and J Zico Kolter (2021). "Orthogonalizing Convolutional Layers with the Cayley Transform". In: *International Conference on Learning Representations*.

Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama (2018). "Lipschitz-Margin Training: Scalable Certification of Perturbation Invariance for Deep Neural Networks". In: *Advances in Neural Information Processing Systems*.

Aladin Virmaux and Kevin Scaman (2018). "Lipschitz Regularity of Deep Neural Networks: Analysis and Efficient Estimation". In: *Advances in Neural Information Processing Systems*.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon (2018). "Towards Fast Computation of Certified Robustness for ReLU Networks". In: *International Conference on Machine Learning*.

Eric Wong and J. Zico Kolter (2018). "Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope". In: *International Conference on Machine Learning*.

Eric Wong, Frank R. Schmidt, Jan Hendrik Metzen, and J. Zico Kolter (2018). "Scaling Provable Adversarial Defenses". In: *Advances in Neural Information Processing Systems*.

Kai Y. Xiao, Vincent Tjeng, Nur Muhammad (Mahi) Shafiullah, and Aleksander Madry (2019). "Training for Faster Adversarial Robustness Verification via Inducing ReLU Stability". In: *International Conference on Learning Representations*.

Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh (2020). "Towards Stable and Efficient Training of Verifiably Robust Neural Networks". In: *International Conference on Learning Representations*.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel (2018). "Efficient Neural Network Robustness Certification with General Activation Functions". In: *Advances in Neural Information Processing Systems* 31.

Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh (2019). "Recurjac: An Efficient Recursive Algorithm for Bounding Jacobian Matrix of Neural Networks and its Applications". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Table 3.2: Comparison to other certified training algorithms. Best numbers are highlighted in bold.

Method	Model	Clean (%)	PGD (%)	Certified(%)
MNIST ($\epsilon = 1.58$)				
Standard	4C3F	99.0	45.4	0.0
LMT (Tsuzuku et al., 2018)	4C3F	86.5	53.6	40.5
CAP (Wong and Kolter, 2018)	4C3F	88.1	67.9	44.5
CROWN-IBP* (Zhang et al., 2020)	4C3F	82.3	80.4	41.3
GloRo (Leino et al., 2021)	4C3F	92.9	68.9	50.1
Local-Lip-G (ours)	4C3F	**96.3**	**78.2**	**55.8**
BCP (Lee et al., 2020)	4C3F	92.4	65.8	47.9
Local-Lip-B (ours)	4C3F	93.0	66.7	48.7
CIFAR-10 ($\epsilon = 36/255$)				
Standard	4C3F	85.3	41.2	0.0
IBP (Gowal et al., 2018)	4C3F	34.5	31.8	24.4
LMT (Tsuzuku et al., 2018)	4C3F	56.5	49.8	37.2
CAP (Wong and Kolter, 2018)	4C3F	60.1	55.7	50.3
CROWN-IBP* (Zhang et al., 2020)	4C3F	54.2	52.7	41.9
ReLU-Stability† (Xiao et al., 2019)	4C3F	57.4	52.4	51.1
GloRo (Leino et al., 2021)	4C3F	73.2	66.3	49.0
Local-Lip-G (ours)	4C3F	**75.7**	**68.6**	49.7
BCP (Lee et al., 2020)	4C3F	64.4	59.4	50.0
Local-Lip-B (ours)	4C3F	70.1	64.2	**53.5**
Standard	6C2F	87.5	32.5	0.0
IBP (Gowal et al., 2018)	6C2F	33.0	31.1	23.4
LMT (Tsuzuku et al., 2018)	6C2F	63.1	58.3	38.1
CAP (Wong and Kolter, 2018)	6C2F	60.1	56.2	50.9
CROWN-IBP* (Zhang et al., 2020)	6C2F	53.7	52.2	41.9
GloRo (Leino et al., 2021)	6C2F	70.7	63.8	49.3
Local-Lip-G (ours)	6C2F	**76.4**	**69.2**	51.3
BCP (Lee et al., 2020)	6C2F	65.7	60.8	51.3
Local-Lip-B (ours)	6C2F	70.7	64.8	**54.3**
GloRo + MaxMin (Leino et al., 2021)	6C2F	77.0	69.2	58.4
Caylay + MaxMin (Trockman and Kolter, 2021)	6C2F	75.3	67.7	59.2
Local-Lip-B + MaxMin (ours)	6C2F	**77.4**	**70.4**	**60.7**
Tiny-Imagenet ($\epsilon = 36/255$)				
Standard	7C1F	35.9	19.4	0.0
GloRo (Leino et al., 2021)	7C1F	31.3	28.2	13.2
Local-Lip-G (ours)	8C2F	**37.4**	**34.2**	13.2
BCP (Lee et al., 2020)	8C2F	28.7	26.6	20.0
Local-Lip-B (ours)	8C2F	30.8	28.4	**20.7**
Gloro + MaxMin (Leino et al., 2021)	8C2F	35.5	32.3	22.4
Local-Lip-B + MaxMin (ours)	8C2F	**36.9**	**33.3**	**23.4**

* CROWN-IBP was originally designed for ℓ_∞ norm certified defense but its released code also supports ℓ_2 training. We use the same hyperparameters as ℓ_∞ training setting.

† (Xiao et al., 2019) was designed for ℓ_∞ norm with a MIP verifier. We extend it to the ℓ_2 norm setting and verify its robustness using the SOTA alpha-beta-CROWN verifier (see Section B.3).

Chapter 4

CERTIFIABLY ROBUST FORWARD INVARIANCE IN NEURAL ODES

Yujia Huang, Ivan Dario Jimenez Rodriguez, Huan Zhang, Yuanyuan Shi, and Yisong Yue (2023). "FI-ODE: Certified and Robust Forward Invariance in Neural ODEs". In: *arXiv preprint arXiv:2210.16940*. URL: https://arxiv.org/abs/2210.16940.

4.1 Introduction

We study the problem of training neural networks with certifiable performance guarantees. Example performance criteria include safety in control (Jin et al., 2020), and adversarial robustness in classification (Wong and Kolter, 2018; Raghunathan et al., 2018; Cohen et al., 2019), where even impressive empirical robustness often fails under unforeseen stronger attacks (Athalye et al., 2018). As such, having formal performance certificates can be valuable when deploying neural networks in high-stakes real-world settings.

In this paper, we are interested in performance criteria characterized by a property called forward invariance. Forward invariance has been extensively used in control theory to certify dynamical systems for safety (Ames et al., 2016) and robustness under adversarial perturbations (Khalil et al., 1996). To use this concept for machine learning, we focus on the Neural ODE (NODE) function class (Haber and Ruthotto, 2017; E, 2017; Chen et al., 2018), which is a natural starting point for incorporating control-theoretic tools (cf. (Yan et al., 2020; Kang et al., 2021; Liu et al., 2020; Jimenez Rodriguez et al., 2022a)). Forward invariance guarantees that NODE trajectories never leave a specified set, which can be translated into various robust safety guarantees. Given the increasing interest in using NODE policies for robotic control (Böttcher and Asikis, 2022; Lin et al., 2021), including those that are forward invariant (Jimenez Rodriguez et al., 2022b), having certified NODE controllers will become important as those methods are more widely adopted.

Our Contributions. We present FI-ODE, a general approach for training certifiably robust forward invariant NODEs.[1] Our approach is based on defining forward

[1] Note that training certifiably forward invariant NODEs even in the non-robust setting itself is a contribution.

invariance using sub-level sets of Lyapunov functions. One can train a NODE such that a task-specific cost function (e.g., state-based cost in continuous control, or cross-entropy loss in image classification) becomes the Lyapunov function for the ODE. We train with an adaptation of Lyapunov training (Jimenez Rodriguez et al., 2022a) that focuses on states that are crucial for certifying robust forward invariance. To make certification practical, we constrain the hidden states of a NODE to evolve on a compact set by projecting the dynamics of NODE to satisfy certain barrier conditions. We provably verify our method through a combination of efficient sampling and a new interval propagation technique compatible with optimization layers.

We evaluate using a canonical unstable nonlinear system (planar segway). We demonstrate certified robust forward invariance of the induced region of attraction, which to our knowledge is the first NODE policy with such non-vacuous certified guarantees. To show generality, we also evaluate on image classification, and show superior ℓ_2 certified robustness versus other certifiably robust ODE-based models. Our code is available at `https://github.com/yjhuangcd/FI-ODE.git`.

4.2 Preliminaries

Neural ODEs. We consider the following Neural ODE (NODE) model class, where $\boldsymbol{x}$ are the inputs to the dynamics, and $\boldsymbol{\eta} \in \mathcal{H} \subset \mathbb{R}^n$ are the states of the NODE ($\mathcal{H}$ is compact and connected). Let $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^l$ denote the parameters of the learned model. In general, we assume the overparameterized setting, where $\boldsymbol{\theta}$ is expressive enough to fit the dynamics.

$$\boldsymbol{\eta}(0) = \boldsymbol{\eta}_0, \qquad \text{(initial condition)} \tag{4.1a}$$

$$\frac{d\boldsymbol{\eta}}{dt} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}(t), \boldsymbol{x}) \quad \text{(continuum of hidden layers).} \tag{4.1b}$$

An important setting for NODEs is continuous control, where we can explicitly compose the known dynamics of the physical system with a neural network controller parameterized by $\boldsymbol{\theta}$. The closed-loop system is denoted by $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}(t), \boldsymbol{x})$, where $\boldsymbol{\theta}$ is the neural controller and $\boldsymbol{x}$ are the system parameters. Other settings include image classification, where $\boldsymbol{x}$ are the images, and we evolve the system over $t \in [0, T]$ to get the final prediction $\boldsymbol{\eta}(T)$.[2]

[2]In this setting, one can think of a NODE as a ResNet (He et al., 2016) with a continuum of hidden layers.

Forward Invariance & Robust Forward Invariance. Forward Invariance refers to sets of states of a dynamical system (e.g., Equation (4.1b)) where the system can enter but never leave. Formally:

Definition 4.2.1 (Forward Invariance)**.** A set $\mathcal{S} \subseteq \mathcal{H}$ is forward invariant with respect to the system (Equation (4.1b)) if $\boldsymbol{\eta}(t) \in \mathcal{S} \Rightarrow \boldsymbol{\eta}(t') \in \mathcal{S}, \forall t' \geq t$.

Forward invariance can be applied generally in NODEs: we can choose the dynamics in Equation (4.1b) to render almost any set we choose forward invariant. For instance, in control we often want to keep the states of the system within a safe set, while in classification we will be concerned with the set of states that produce a correct classification. Mathematically, these settings can be captured by shaping the ODE dynamics $\boldsymbol{f_\theta}$ to achieve forward invariance within a specified set (i.e., training the ODE to satisfy Definition 4.2.1 for some specified set $\mathcal{S}$).

Definition 4.2.2 (Robust Forward Invariance)**.** A set $\mathcal{S} \subseteq \mathcal{H}$ is robust forward invariant with respect to $\boldsymbol{x}$ if $\mathcal{S}$ is forward invariant with respect to the system $\boldsymbol{f_\theta}(\boldsymbol{\eta}(t), \boldsymbol{x} + \boldsymbol{\epsilon}), \forall \boldsymbol{\epsilon} \in \mathbb{R}^n$ with $\|\boldsymbol{\epsilon}\| \leq \bar{\epsilon}$.

Robust forward invariance is attractive when one seeks performance guarantees under input perturbations. Here, we consider norm-bounded perturbations. In control, when there are mis-specifications for system parameters, we still hope the controller to be able to keep the system safe. In classification, we would want the system to classify correctly despite noisy inputs.

Trajectory-wise versus Point-wise Certification Analysis. Figure 4.1 depicts two ways of certifying forward invariance. On the left, we consider entire trajectories that result from running the ODE (i.e., running the forward pass) and determine forward invariance by checking whether the *trajectories* leave the target set. Such trajectory-level analyses are computationally expensive due to running ODE integration to generate trajectories. This approach also poses a challenge for verification since trajectories can only be integrated for finite time T and the dynamics may be close to leaving the set shortly thereafter ($T + \varepsilon$), which translates into vulnerability to perturbations.

An alternative approach, depicted on Figure 4.1(right), relies on point-wise conditions: we look at the dynamics *point-wise* over the state-space and infer whether the set within the yellow line is forward invariant. Here, robust certification can be

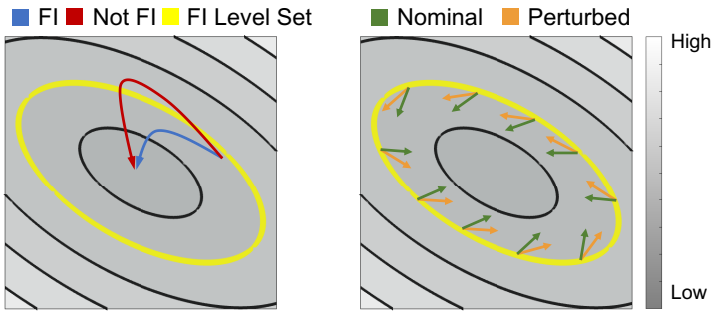


Figure 4.1: **Depicting trajectories (Left) and dynamics (Right) of the state-space of a NODE.** The contours show a quadratic potential, and the yellow-line is the target sublevel set. Left: trajectories that violate (red) or satisfy (blue) forward invariance. Right: flow field (dynamics) of the NODE, under both nominal and perturbed inputs. The perturbed flow field still satisfies forward invariance, implying robust forward invariance.

significantly easier because we only need to verify that the perturbed dynamics are point-wise still pointing in the right direction, rather than analyzing the perturbed dynamics over an entire trajectory (i.e., we do not need to do ODE integration).

Lyapunov Functions & Sublevel Sets. As discussed further in Section 4.3, we use Lyapunov potential functions from control theory to define sets to render forward invariant. A potential function $V : \mathcal{H} \to \mathbb{R}_{\geq 0}$ is a Lyapunov function for the ODE if for all reachable states η we have:

$$\dot{V} \equiv \frac{d}{dt}V(\boldsymbol{\eta}(t)) \equiv \frac{\partial V}{\partial \boldsymbol{\eta}}^{\top} \boldsymbol{f_\theta}(\boldsymbol{\eta}, \boldsymbol{x}) \leq 0. \tag{4.2}$$

Intuitively, Equation (4.2) means that the dynamics of the ODE within the states $\mathcal{H}$ are always flowing in the direction that reduces V, i.e., Equation (4.2) establishes a contraction condition on ODE. Note that for the system to be strictly contracting, the RHS of Equation (4.2) needs to be strictly negative.

Since V is always decreasing in time, we can use it to define a forward invariant set (Definition 4.2.1): $V(\boldsymbol{\eta}) \leq c$ for some constant c, which is known as a Lyapunov sublevel set. Once a state enters a Lyapunov sublevel set it remains there for all time. The potential function depicted in Figure 4.1 can be viewed as a Lyapunov function, and the yellow line the boundary of the corresponding sublevel set. At training time, one would specify a desired forward invariance condition using a potential function V and threshold c, and optimize the NODE to satisfy the forward invariance condition.

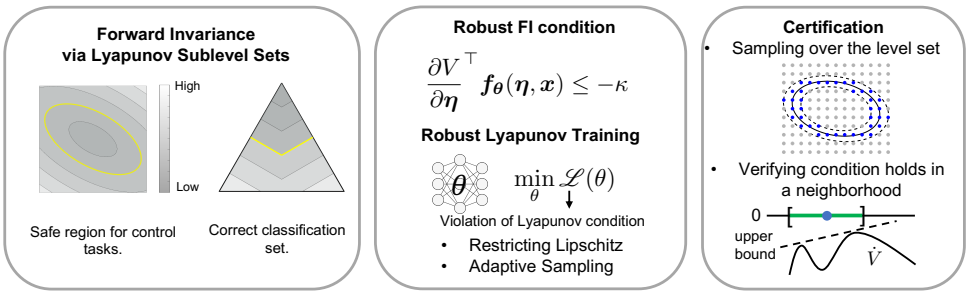


Figure 4.2: **Overview of our FI-ODE framework.** We first pick a Lyapunov function based on the shape of the forward invariant set: the boundaries of the Lyapunov sublevel sets are parallel to the boundary of the forward invariant set. Then we show that robust forward invariance implies robust control and classification. We train the dynamics to satisfy robust FI conditions via robust Lyapunov training. To certify the forward invariance property, we sample points on the boundary of the forward invariant set and verify conditions hold everywhere on the boundary.

4.3 FI-ODE: Robust Forward Invariance for Neural ODEs

We now present our FI-ODE framework to enforce forward invariance on NODEs (Figure 5.1). We define forward invariance using Lyapunov sublevel sets (Section 4.3), and show that robust forward invariance implies robust control and classification (Section 4.3) To enforce forward invariance, we first train to encourage the Lyapunov conditions to hold on the boundary of the target Lyapunov sublevel set (e.g., yellow line in Figure 4.1), and then verify. We develop a robust Lyapunov training algorithm (Section 4.3) that extends the LyaNet framework (Jimenez Rodriguez et al., 2022a) to enable efficiently training NODEs that provably satisfy forward invariance. Finally, we develop certification tools to verify the Lyapunov conditions everywhere in the region of interest (Section 4.3).

Forward Invariance via Lyapunov Sublevel Sets

We first define the set that we would like to render forward invariant, and then choose a Lyapunov function whose level sets are parallel to the boundary of the set.[3] For our main application in control, we define the forward invariant set to be a region around an equilibrium point (i.e., straying far from the equilibrium point can be unsafe), and use the standard quadratic Lyapunov function:

$$V(\boldsymbol{\eta}) = \boldsymbol{\eta}^\top P \boldsymbol{\eta}, \tag{4.3}$$

[3]Usually, Lyapunov stability uses a potential function to prove the stability of a given dynamical system. In our setting, the potential function is pre-defined to be positive definite, and we find a dynamical system (e.g., by training a Neural ODE) that is stable with respect to this potential function (i.e., making this potential function a Lyapunov function). This is possible because the NODEs are typically overparameterized.

where P is a (learnable) positive definite matrix, and assuming WLOG that the equilibrium point is at the origin. The forward invariant set has the form $\mathcal{S} = \{\boldsymbol{\eta} | V(\boldsymbol{\eta}) \leq c\}$, for $c > 0$. The level sets of this quadratic Lyapunov function are shown in Figure 4.1. The boundary is then $\mathcal{D} = \partial\mathcal{S} = \{\boldsymbol{\eta} | V(\boldsymbol{\eta}) = c\}$. This forward invariance condition is commonly used in safety-critical control (Ames et al., 2019).

We also explore an application to multi-class classification. Here, we define the forward invariant set to be the correct classification region. For an input $\boldsymbol{x}$ with label y, the output of a NODE after integrating for T time is $\boldsymbol{\eta}(T)$. The NODE correctly classifies $\boldsymbol{x}$ if $y = \arg\max \boldsymbol{\eta}(T)$. Then the correct classification region for class y is $\mathcal{S}_y = \{\boldsymbol{\eta} | \boldsymbol{\eta} \in \triangle, y = \arg\max \boldsymbol{\eta}\}$ (Figure 5.1, left panel) where $\triangle$ stands for the n-class probability simplex: $\{\boldsymbol{\eta} \in \mathbb{R}^n | \sum_{i=1}^n \boldsymbol{\eta}_i = 1, \boldsymbol{\eta}_i \geq 0\}$. The boundary of this set is known to be the decision boundary for class y: $\mathcal{D}_y = \{\boldsymbol{\eta} \in \triangle | \boldsymbol{\eta}_y = \max_{i \neq y} \boldsymbol{\eta}_i\}$. We define a Lyapunov function whose level sets are parallel to the decision boundary:

$$V_y(\boldsymbol{\eta}) = 1 - (\boldsymbol{\eta}_y - \max_{i \neq y} \boldsymbol{\eta}_i)\,. \tag{4.4}$$

We can check that V_y is positive definite: since $0 \leq \eta_i \leq 1$ for all i, we have $V_y \geq 0$. In addition, $V_y = 0$ only when $\eta_y = 1$ and $\eta_i = 0$ for $i \neq y$. For the simplicity of notations, we use V to refer to the Lyapunov function, but note that the Lyapunov function for classification depends on class y.

Robust Forward Invariance for Robust Control and Classification

A NODE satisfies robust forward invariance if the forward invariance condition holds despite (norm-bounded) perturbations on the dynamics (e.g., due to perturbed inputs). Our framework uses $\boldsymbol{x}$ for *system parameters* in control, and for *input images* in classification. To ensure robust forward invariance for perturbed $\boldsymbol{x}$, the dynamics for the perturbed input $\boldsymbol{f_\theta}(\boldsymbol{\eta}, \boldsymbol{x} + \boldsymbol{\epsilon})$ needs to satisfy the standard forward invariance condition in (4.2) (as informally depicted in Figure 4.1, right). In other words, the condition in (4.2) needs to hold in a neighborhood of $\boldsymbol{x}$ for robust control or classification. Thanks to the Lipschitz continuity of the Lyapunov function V and the dynamics $\boldsymbol{f_\theta}$, this can be achieved by a more strict condition than (4.2) on the dynamics (Theorem 4.3.1).

Theorem 4.3.1 (Robust Forward Invariance)**.** *Consider the dynamical system in Equations* (4.1a) *and* (4.1b)*, the set $\mathcal{S}$ will be robust forward invariant with respect to $\boldsymbol{x}$ if the following conditions hold:*

$$\frac{\partial V}{\partial \boldsymbol{\eta}}^\top \boldsymbol{f_\theta}(\boldsymbol{\eta}, \boldsymbol{x}) \leq -\bar{\epsilon} L_V L_f^x, \quad \forall \boldsymbol{\eta} \in \partial\mathcal{S} \tag{4.5}$$

where $\bar{\epsilon}$ is the perturbation magnitude on $\boldsymbol{x}$ (i.e., $\|\boldsymbol{\epsilon}\| \leq \bar{\epsilon}$), $\partial\mathcal{S}$ is the boundary of $\mathcal{S}$, L_V is the Lipschitz constant of V and L_f^x is the Lipschitz constant of the dynamics with respect to x.

Remark (Implications)**.** *With $\mathcal{S}$ defined as in Section 4.3, if the dynamics satisfy* (4.5)*, then we have a robust controller that always keeps the system in the desired region despite perturbed system parameters and inputs.*

Remark (Non-Robust Variant)**.** *The non-robust version of Theorem 4.3.1 is where the RHS of Equation* (4.5) *is 0 instead of $-\bar{\epsilon}L_V L_f^x$. I.e., Equation* (4.5) *need not be a strictly contracting condition.*

Robust Lyapunov Training

We now present our robust Lyapunov training approach to satisfy the conditions in Theorem 4.3.1 (Algorithm 3). Our method extends the LyaNet framework (Jimenez Rodriguez et al., 2022a) in two ways: 1) restricting the Lipschitz constant of the NODE with respect to the input; and 2) adaptive sampling to focus learning on the states necessary for forward invariance certification.

Training loss. Our training loss encourages the dynamics to satisfy the conditions in Theorem 4.3.1. Specifically, we use a modified Monte Carlo Lyapunov loss from (Jimenez Rodriguez et al., 2022a):

$$\mathscr{L}(\boldsymbol{\theta}) \approx \mathop{\mathbb{E}}_{\boldsymbol{\eta}\sim\mu(\mathcal{H})}\left[\max\left\{0, \frac{\partial V}{\partial \boldsymbol{\eta}}^{\top} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x}) + \kappa(V(\boldsymbol{\eta}))\right\}\right], \tag{4.6}$$

which can be interpreted as a hinge-like loss on the Lyapunov contraction condition for each state η of the NODE (i.e., the loss encourages $(\partial V/\partial \boldsymbol{\eta})^{\top}\boldsymbol{f} \leq -\kappa(V(\boldsymbol{\eta})) < 0$ for some non-negative non-decreasing function κ). Intuitively, if the loss in Equation (4.6) is 0 for some given $\boldsymbol{x}$, then we know that the contraction condition is satisfied with the RHS being $\kappa(V(\boldsymbol{\eta}))$. As long as $\kappa(V(\boldsymbol{\eta})) \geq \bar{\epsilon}L_V L_f^x$, then Theorem 4.3.1 holds. If instead $\kappa(V(\boldsymbol{\eta})) \geq 0$, then the non-robust variant holds.

Restricting the Lipschitz constant. To obtain a non-vacuous guarantee from Theorem 4.3.1, we need to restrict the Lipschitz of $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x})$ with respect to both $\boldsymbol{\eta}$ and $\boldsymbol{x}$. For image classification, we can estimate L_f^x easily by the product of matrix norms of the weight matrices in the neural network (Tsuzuku et al., 2018). Then we certify that condition Equation (4.5) holds for the clean image $\boldsymbol{x}$. For control problems, since the dynamics of the physical system is usually known, the

Algorithm 3 Robust Lyapunov Training

Require: Lyapunov function V, Sampling scheduler, dataset $\mathcal{D}$, hinge-like function κ.

Initialize: Model parameters $\boldsymbol{\theta}$, Lyapunov parameters P and system parameters $\boldsymbol{x}$ (for control).
for $i = 1 : M$ **do**
  ▷ Sample $\boldsymbol{\eta}$ based on the training progress and the level sets of V:
    $\boldsymbol{\eta} \sim$ `Sampling_scheduler` (i, V)
  ▷ For control, find adversarial samples of $\boldsymbol{x}$ and $\boldsymbol{\eta}$
    For classification, sample $(\boldsymbol{x}, y) \sim \mathcal{D}$
  ▷ Update model parameters to minimize Lyapunov loss $\mathscr{L}(\boldsymbol{\theta})$ (Equation (4.6)).
  $P \leftarrow P - \beta' \nabla_P \mathscr{L}(\boldsymbol{\theta})$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \mathscr{L}(\boldsymbol{\theta})$
end for

Ensure: $\boldsymbol{\theta}$

closed loop dynamics is not purely parameterized by neural networks and it is not straightforward to estimate L_f^x. Therefore, instead of directly certifying condition Equation (4.5), we certify the LHS of it to be smaller than 0 for $\boldsymbol{\eta} \in \partial\mathcal{S}$ and all $\boldsymbol{x}$ within the perturbation range (not only on the nominal parameter). We use adversarial training to make $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x})$ smooth with respect to both $\boldsymbol{\eta}$ and $\boldsymbol{x}$, and in fact certifiable.

Adaptive sampling. To minimize the Lyapunov loss (Equation (4.6)), we need to choose a sampling distribution μ. A simple choice is uniform (as was done in (Jimenez Rodriguez et al., 2022a)), but that may require an intractable number of samples to guarantee minimizing Equation (4.6) everywhere in the state space. We address this challenge with an adaptive sampling strategy that focuses training samples on the region of the state space necessary for forward invariance: the boundary of the Lyapunov sub-level set. For control problems, since we jointly learn matrix P in the Lyapunov function, the shape of its level set changes during training, and the sampled points change accordingly. For classification, we switch from uniform sampling in the simplex to sampling only within the forward invariant set, with the switching time being a hyper-parameter of the sampling scheduler.

Certification

In the previous section, we minimize the empirical Lyapunov loss (Equation (4.6)) on some finite set of samples to encourage the dynamics to satisfy conditions in Theorem 4.3.1. However, zero empirical Lyapunov loss on a finite sample is not

necessarily a certificate that the conditions hold everywhere on the boundary of the forward invariant. This section develops tools to certify the forward invariance conditions hold *everywhere* on the boundary of the safe set.

Certification procedures. The certification procedures are as follows: 1) sample points on the boundary of the forward invariant set (blue dots in Figure 4.3), and check Lyapunov condition holds on all the sampled points; 2) verify the condition holds in a small neighborhood around those points.

Procedure 1: Sampling techniques. Rigorous certification is challenging because it requires the set of samples and their neighborhoods to cover the whole boundary of the forward invariant set (not guaranteed by random sampling). We construct a set for the quadratic Lyapunov function (Equation (4.3)) and the classification Lyapunov function (Equation (4.4)) respectively, and show that the proposed set can cover the level set of the corresponding Lyapunov function in Theorem 4.3.2 below, i.e., for any point on the level set (red dot in Figure 4.3), there exists a sampled point nearby (blue dot).

To sample on the level set of the quadratic Lyapunov function $\mathcal{D} = \{\boldsymbol{\eta} \in \mathbb{R}^n | \boldsymbol{\eta}^\top P \boldsymbol{\eta} = c\}$, we first create a uniform grid $\mathcal{G}$ (with spacing r) in the ambient space that covers the Lyapunov level set. We pick r to be at most $\sqrt{\frac{c}{\lambda_1}}$, where λ_1 is the maximum eigenvalue of P. Then we do rejection sampling to keep the points that are close to the Lyapunov level set via $\mathcal{G} \cap \mathcal{B}$ for $\mathcal{B}$ defined below:

$$\mathcal{B} = \{\boldsymbol{\eta} | \underline{c} \leq \boldsymbol{\eta}^\top P \boldsymbol{\eta} \leq \bar{c}\} \tag{4.7}$$

where $\underline{c} = (\sqrt{c} - \frac{\sqrt{n}}{2} r \sqrt{\lambda_1})^2$ and $\bar{c} = (\sqrt{c} + \frac{\sqrt{n}}{2} r \sqrt{\lambda_1})^2$. We show that $\mathcal{B} \cap \mathcal{G}$ (points inside dashed lines in Figure 4.3, Left) covers the c-level set of the quadratic Lyapunov function (Theorem 4.3.2 (a)).

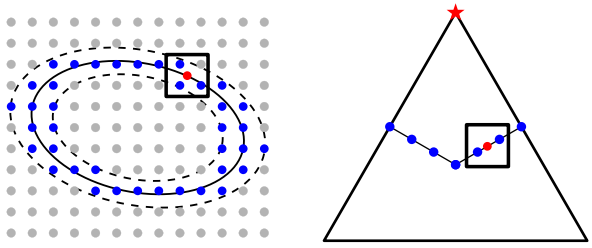


Figure 4.3: **Sampling to cover the level sets of the Lyapunov functions.**

To sample on the 1-level set of the n-class classification Lyapunov function (the decision boundary), we consider the following set $\tilde{S}_y$ (Figure 4.3, Right) (Theorem 4.3.2 (b)):

$$\tilde{S}_y = \{\tilde{\boldsymbol{s}} \in \mathbb{R}^n | \tilde{\boldsymbol{s}} = \frac{\boldsymbol{s}}{N}, \boldsymbol{s} \in S_y\} \tag{4.8}$$

where $S_y = \{\boldsymbol{s} \in \mathbb{Z}^n | \sum_{i=1}^n \boldsymbol{s}_i = N, \boldsymbol{s}_y = \max_{i \neq y} \boldsymbol{s}_i, \boldsymbol{s}_i \geq 0, \forall i = 1, ..., n\}$, and N represents sample density, and needs to be a positive even integer and $N \not\equiv 1 \pmod{n}$.

Theorem 4.3.2. (Sampling on the boundary of a FI set).

- **(a)** For any $\boldsymbol{\eta} \in \mathcal{D}$, there exist an $\boldsymbol{s} \in \{\mathcal{B} \cap \mathcal{G}\}$ such that $|\boldsymbol{\eta}_i - \boldsymbol{s}_i| \leq \frac{r}{2}$ for all $i = 1, ..., n$.
- **(b)** For any $\boldsymbol{\eta} \in \mathcal{D}_y$, there exists an $\tilde{\boldsymbol{s}} \in \tilde{S}_y$ such that $|\boldsymbol{\eta}_i - \tilde{\boldsymbol{s}}_i| \leq \frac{1}{N}$ for all $i = 1, ..., n$.

Procedure 2: Verification in a neighborhood around the sampled points. Since we only sample a finite number of points on the level set, certifying robust forward invariance requires verifying that the condition holds in a small neighborhood around each of the points. We do so by bounding the range of the output given the range of the input. This bound can be obtained by estimating the Lipschitz constant of the LHS of Equation (4.5), and the norm of output difference can be bounded by the norm of the input difference. This is convenient for cases where it is simple to bound the Lipschitz of the Lyapunov function and the dynamics. For instance, for classification problems, the Lipschitz constant of the Lyapunov function (4.4) is $\sqrt{2}$, and the Lipschitz constant of the dynamics with respect to both $\boldsymbol{\eta}$ and $\boldsymbol{x}$ is 1 because we use orthogonal layers in the neural network. For more general Lyapunov functions and dynamics, the Lipschitz bound is often either intractable or vacuous. Instead, we use a popular linear relaxation based verifier CROWN (Zhang et al., 2018) to bound the output of any general computation graph. In the control case, we verify both $\boldsymbol{x}$ and $\boldsymbol{\eta}$ since we want the robustness robustness for a set of perturbations over a set in the state-space.

4.4 Experiments

Our main evaluation is in an application of certified robust forward invariance in nonlinear continuous control (Section 4.4). We also explore the generality of our approach by studying a second application in certified robustness for image classification (Section 4.4).

Certifying Safety for Robust Continuous Control

Setup. We evaluate our framework on a planar segway system, which is a highly unstable nonlinear system whose dynamics is sensitive to its system parameters and therefore hard to train certifiably robust nonlinear controllers (see Appendix C.5 for the details). We train a neural network controller (a 3-layer multi-layer perceptron (MLP)) to keep the system forward invariant within the 0.15-sublevel set of a jointly learned Lyapunov function under $\pm 2\%$ perturbations on each system parameter. This guarantees that the segway will not fall under adversarial system perturbations. We evaluate the its performance under both nominal and adversarial system parameters for 1000 adversarially selected initial states within the safe set. The adversarial parameters and states are optimized jointly via projected gradient descent for 100 steps to maximize violations of the forward invariance condition. We also provide provable certificates using certification approach in Section 4.3.

Results. We compare with two competitive robust control baselines, and perform ablation studies on different training algorithms in Table 4.1. We make three main observations. First, it is difficult to certify even non-robust forward invariance using previous methods, highlighting the need for more advanced methods. Second, our robust Lyapunov training approach (Algorithm 3) is able to train NODE controllers with robust forward invariance certificates. Third, certifying non-robust forward invariance is easier than certifying robust forward invariance. Overall, these results suggest that our approach is able to train nonlinear ODE controllers in non-trivial settings where existing approaches cannot. To our knowledge, this is also the first instance of training NODE policies with such non-vacuous certified guarantees.

Table 4.1: Robustness of controllers trained with different methods. The numbers are the percentage of trajectories that stay within the forward invariant set under the nominal and adversarial system parameters on 1000 adversarially selected initial states. The certificate column indicates whether the (robust) FI property is certified.

Method	**Empirical**		**Certificate**	
	Nominal	**Adv**	**FI**	**Robust FI**
Robust LQR	96.2	92.8	✗	✗
Robust MBP (Donti et al., 2020)	94.3	93.6	✗	✗
Standard Backprop Training	58.0	50.4	✗	✗
Basic Lyapunov Training (Jimenez Rodriguez et al., 2022a)	90.2	52.6	✗	✗
+ Adaptive Sampling	100	68.9	✓	✗
+ Adversarial Training	100	97.8	✓	✗
+ Both (Robust FI-ODE, Ours)	100	100	✓	✓

Visualizations. We show trajectories that start within the safe set (gray ellipse) and the corresponding Lyapunov functions in Figure 4.4. The left shows the trajectories of a certifiably *non-robust* FI controller. While the system is safe 100% under nominal system parameters, it fails for adversarial system parameters. The right shows the trajectories of a certifiably *robust* FI controller. Even under adversarial system parameters, it keeps the trajectories within the safe set 100% of the time.

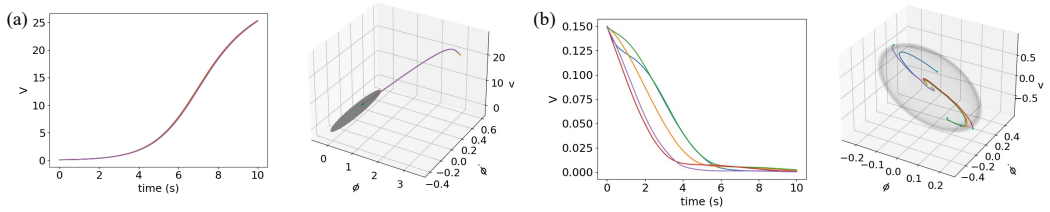


Figure 4.4: **Showing Lyapunov function value V along the trajectories of a planar segway.** The forward invariant set is the 0.15-sublevel set. All trajectories start within the forward invariant set (gray ellipse). Each system parameter are perturbed adversarially within $\pm 2\%$ of their original value. (a) Shows the Lyapunov function values and system trajectories of a certifiably *non-robust* FI controller. (b) Shows the Lyapunov function values and system trajectories of a certifiably *robust* FI controller.

Certified Robustness for Image Classification

We also apply our approach to train certifiably robust NODEs for image classification to explore the generality of the framework. We treat the input image as system parameters, and set all the initial states to be $\boldsymbol{\eta}(0) = \mathbb{1}\frac{1}{n}$. Table 4.2 shows the results. The main metric is certified accuracy, the percentage of test set inputs that are certifiably robust. Our approach achieves the strongest overall certified robustness results compared to prior ODE-based approaches. We also reported clean and adversarial accuracy for references. In addition, the ablation studies shows similar trends as in the robust control experiments: all the learning components: Lyapunov training, Lipschitz restriction and adaptive sampling are needed for good performance, and the model that is trained with all of them (Robust FI-ODE) achieved the highest certified accuracy.

4.5 Related Works

Robust control and robust learning-based control methods. Robust control involves creating feedback controllers for dynamic systems that sustain performance under adverse conditions (Zhou and Doyle, 1998; Başar and Bernhard, 2008), often relying on basic (linear) controllers. Our work learns *nonlinear* controllers

Table 4.2: Evaluating certified robustness for image classification. ϵ is the ℓ_2 norm of the input perturbations. We report the classification accuracy (%) on clean & adversarial inputs, and the percentage of inputs that are certifiably robust (Certified). Semi-MonDeq results are on 100 test images [95% CI in bracket] due to high cost, and other results are on all test images (10,000).

Dataset	Method	ϵ	Clean	Adversarial	Certified
MNIST	Lipschitz-MonDeq (Pabbaraju et al., 2020)	0.1	95.60	94.42	83.09
	Semi-MonDeq (Chen et al., 2021) †	0.1	99 [>94]	99 [>94]	99 [>94]
	Robust FI-ODE (Ours)	0.1	99.35	99.09	95.75
	Lipschitz-MonDeq (Pabbaraju et al., 2020)	0.2	95.60	93.09	50.56
	Robust FI-ODE (Ours)	0.2	99.35	98.83	81.65
CIFAR-10	Lipschitz-MonDeq (Pabbaraju et al., 2020)	0.141	66.66	50.51	<7.37
	NODE w/o Lyapunov training	0.141	69.05	56.94	16.81
	LyaNet (Jimenez Rodriguez et al., 2022a) + Lipschitz restriction	0.141	73.15	64.87	41.43
	LyaNet (Jimenez Rodriguez et al., 2022a) + Sampling scheduler	0.141	82.83	74.81	0
	Robust FI-ODE (Ours)	0.141	78.34	67.45	42.27

parameterized by neural networks, while maintaining the robust forward invariance guarantees. There have been recent works for learning-based control with robustness guarantees, such as focusing on $\mathcal{H}_\infty$ robust control (Abu-Khalaf et al., 2006; Luo et al., 2014; Friedrich and Buss, 2017; Han et al., 2019; Zhang et al., 2020), or linear differential inclusions systems (Donti et al., 2020). In comparison, our framework could be used for general nonlinear systems and norm-bounded input/system parameter perturbations.

Learning Lyapunov functions and controllers for nonlinear control problems. Various studies focus on learning neural network Lyapunov functions, barrier functions, and contraction metrics for nonlinear control (Dawson et al., 2023). For stability or safety certification, Chang et al. (2019) employ SMT solvers (Gao et al., 2013), Jin et al. (2020) use Lipschitz methods, and Dai et al. (2021) apply mixed integer programming. Our approach uses a linear relaxation-based verifier (Zhang et al., 2018), balancing tightness and computational efficiency, to certify nonlinear control policies (unlike the linear policies in Chang et al. (2019) and Jin et al. (2020)) on actual dynamics, contrasting with Dai et al. (2021)'s neural network dynamic approximations.

Verification and Certified robustness of NODEs. Many studies (e.g., Yan et al. (2020), Kang et al. (2021), and Huang et al. (2022)) demonstrate improved empirical robustness of NODEs, yet certifying this robustness is challenging. Prior NODE analyses primarily address reachability: Grunbacher et al. (2021) proposes a stochastic bound on the reachable set of NODEs, while Lopez et al. (2022) computes deterministic reachable set of NODEs via zonotope and polynomial-zonotope based

methods implemented in CORA (Althoff, 2013). However, these methods are limited to low-dimension or linear NODEs. MonDEQ (Winston and Kolter, 2020), akin to implicit ODEs, has seen ℓ_2 robustness certification efforts (Pabbaraju et al., 2020; Chen et al., 2021), but these struggle beyond MNIST. Xiao et al. (2023) propose invariance propagation for stacked NODEs that provides guarantees for output specifications by controller/input synthesis. While their approach focuses more on interpretable causal reasoning of stacked NODEs, our work provides a framework for training and provably certifying general NODEs.

Formal verification of neural networks. Formal verification of neural networks aim to prove or disprove certain specifications of neural networks, and a canonical problem of neural network verification is to bound the output of neural networks given specified input perturbations. Computing the exact bounds is a NP-complete problem (Katz et al., 2017) and can be solved via MIP or SMT solvers (Tjeng et al., 2019; Ehlers, 2017), but they are not scalable and often too expensive for practical usage. In the meanwhile, incomplete neural network verifiers are developed to give sound outer bounds of neural networks (Salman et al., 2019; Dvijotham et al., 2018; Wang et al., 2018; Singh et al., 2019), and bound-propagation-based methods such as CROWN (Zhang et al., 2018) are a popular approach for incomplete verification. Recently, branch-and-bound based approaches (Bunel et al., 2020; Wang et al., 2021; De Palma et al., 2021) are proposed to further enhance the strength of neural network verifiers. Our work utilizes neural network verifiers as a sub-procedure to prove forward invariance of NODEs, and is agnostic to the verification algorithm used. We used CROWN because it is efficient, GPU-accelerated and has high quality implementation (Xu et al., 2020).

References

Murad Abu-Khalaf, Frank L Lewis, and Jie Huang (2006). "Policy Iterations on the Hamilton–Jacobi–Isaacs Equation for $\mathcal{H}_\infty$ State Feedback Control With Input Saturation". In: *IEEE Transactions on Automatic Control* 51.12, pp. 1989–1995.

Matthias Althoff (2013). "Reachability Analysis of Nonlinear Systems using Conservative Polynomialization and Non-convex Sets". In: *Proceedings of the 16th international conference on hybrid systems: computation and control*, pp. 173–182.

Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada (2019). "Control Barrier Functions: Theory and

Applications". In: *2019 18th European control conference (ECC)*. IEEE, pp. 3420–3431.

Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada (2016). "Control Barrier Function Based Quadratic Programs for Safety Critical Systems". In: *IEEE Transactions on Automatic Control*.

Anish Athalye, Nicholas Carlini, and David A. Wagner (2018). "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples". In: *International Conference on Machine Learning*.

Tamer Başar and Pierre Bernhard (2008). *H-infinity Optimal Control and Related Minimax Design Problems: a Dynamic Game Approach*. Springer Science & Business Media.

Lucas Böttcher and Thomas Asikis (2022). "Near-optimal Control of Dynamical Systems with Neural Ordinary Differential Equations". In: *Machine Learning: Science and Technology* 3.4, p. 045004.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar (2020). "Branch and bound for piecewise linear neural network verification". In: *Journal of Machine Learning Research* 21.42, pp. 1–39.

Ya-Chien Chang, Nima Roohi, and Sicun Gao (2019). "Neural Lyapunov Control". In: *Advances in Neural Information Processing Systems* 32.

Ricky T.Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud (2018). "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems*.

Tong Chen, Jean B Lasserre, Victor Magron, and Edouard Pauwels (2021). "Semialgebraic Representation of Monotone Deep Equilibrium Models and Applications to Certification". In: *Advances in Neural Information Processing Systems* 34, pp. 27146–27159.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter (2019). "Certified Adversarial Robustness via Randomized Smoothing". In: *International Conference on Machine Learning*.

Hongkai Dai, Benoit Landry, Lujie Yang, Marco Pavone, and Russ Tedrake (2021). "Lyapunov-stable Neural-network Control". In: *Robotics: Science and Systems (RSS)*.

Charles Dawson, Sicun Gao, and Chuchu Fan (2023). "Safe Control with Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction Methods for Robotics and Control". In: *IEEE Transactions on Robotics*.

Alessandro De Palma, Harkirat Singh Behl, Rudy Bunel, Philip H. S. Torr, and M. Pawan Kumar (2021). "Scaling the Convex Barrier with Active Sets". In: *International Conference on Learning Representations*.

Priya L Donti, Melrose Roderick, Mahyar Fazlyab, and J Zico Kolter (2020). "Enforcing Robust Control Guarantees within Neural Network Policies". In: *International Conference on Learning Representations*.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli (2018). "A Dual Approach to Scalable Verification of Deep Networks". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Weinan E (2017). "A Proposal on Machine Learning via Dynamical Systems". In: *Communications in Mathematics and Statistics* 5.1, pp. 1–11.

Ruediger Ehlers (2017). "Formal Verification of Piece-wise Linear Feed-forward Neural Networks". In: *International Symposium on Automated Technology for Verification and Analysis (ATVA)*.

Stefan R. Friedrich and Martin Buss (2017). "A Robust Stability Approach to Robot Reinforcement Learning based on a Parameterization of Stabilizing Controllers". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3365–3372.

Sicun Gao, Soonho Kong, and Edmund M. Clarke (2013). "dReal: An SMT Solver for Nonlinear Theories over the Reals". In: *International Conference on Automated Deduction*. Springer, pp. 208–214.

Sophie Grunbacher, Ramin Hasani, Mathias Lechner, Jacek Cyranka, Scott A Smolka, and Radu Grosu (2021). "On the Verification of Neural ODEs with Stochastic Guarantees". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Eldad Haber and Lars Ruthotto (Dec. 2017). "Stable Architectures for Deep Neural Networks". In: *Inverse Problems* 34.1, p. 014004.

Minghao Han, Yuan Tian, Lixian Zhang, Jun Wang, and Wei Pan (2019). "$\mathcal{H}_\infty$ Model-free Reinforcement Learning with Robust Stability Guarantee". In: *arXiv preprint:1911.02875*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Yifei Huang, Yaodong Yu, Hongyang Zhang, Yi Ma, and Yuan Yao (2022). "Adversarial Robustness of Stabilized Neural ODE Might be from Obfuscated Gradients". In: *Mathematical and Scientific Machine Learning*.

Ivan Dario Jimenez Rodriguez, Aaron D Ames, and Yisong Yue (2022a). "LyaNet: A Lyapunov Framework for Training Neural ODEs". In: *International Conference on Machine Learning*.

Ivan Dario Jimenez Rodriguez, Noel Csomay-Shanklin, Yisong Yue, and Aaron D Ames (2022b). "Neural Gaits: Learning Bipedal Locomotion via Control Barrier Functions and Zero Dynamics Policies". In: *Conference on Learning for Dynamics and Control (L4DC)*.

Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou (2020). "Neural Certificates for Safe Control Policies". In: *arXiv preprint arXiv:2006.08465*.

Qiyu Kang, Yang Song, Qinxu Ding, and Wee Peng Tay (2021). "Stable Neural ODE with Lyapunov-Stable Equilibrium Points for Defending against Adversarial Attacks". In: *Advances in Neural Information Processing Systems*.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer (2017). "Reluplex: An efficient SMT Solver for Verifying Deep Neural Networks". In: *International Conference on Computer Aided Verification (CAV)*.

Islam Khalil, JC Doyle, and K Glover (1996). *Robust and Optimal Control*. Prentice Hall, New Jersey.

HaoChih Lin, Baopu Li, Xin Zhou, Jiankun Wang, and Max Q-H Meng (2021). "No Need For Interactions: Robust Model-based Imitation Learning using Neural ODE". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 11088–11094.

Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh (2020). "How Does Noise Help Robustness? Explanation and Exploration under the Neural SDE Framework". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (IEEE / CVF Computer Vision and Pattern Recognition Conference)*, pp. 282–290.

Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton, and Taylor T Johnson (2022). "Reachability Analysis of a General Class of Neural Ordinary Differential Equations". In: *arXiv preprint arXiv:2207.06531*.

Biao Luo, Huai-Ning Wu, and Tingwen Huang (2014). "Off-policy Reinforcement Learning for $\mathcal{H}_\infty$ Control Design". In: *IEEE transactions on cybernetics* 45.1, pp. 65–76.

Chirag Pabbaraju, Ezra Winston, and J Zico Kolter (2020). "Estimating Lipschitz Constants of Monotone Deep Equilibrium Models". In: *International Conference on Learning Representations*.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang (2018). "Certified Defenses against Adversarial Examples". In: *International Conference on Learning Representations*.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang (2019). "A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks". In: *Advances in Neural Information Processing Systems*.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev (2019). "An Abstract Domain for Certifying Neural Networks". In: *Proceedings of the ACM on Programming Languages (POPL)*.

Vincent Tjeng, Kai Xiao, and Russ Tedrake (2019). "Evaluating Robustness of Neural Networks with Mixed Integer Programming". In: *International Conference on Learning Representations*.

Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama (2018). “Lipschitz-Margin Training: Scalable Certification of Perturbation Invariance for Deep Neural Networks”. In: *Advances in Neural Information Processing Systems*.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana (2018). “Efficient Formal Safety Analysis of Neural Networks”. In: *Advances in Neural Information Processing Systems*.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter (2021). “Beta-CROWN: Efficient Bound Propagation with per-neuron Split Constraints for Neural Network Robustness Verification”. In: *Advances in Neural Information Processing Systems*.

Ezra Winston and J Zico Kolter (2020). “Monotone Operator Equilibrium Networks”. In: *Advances in Neural Information Processing Systems* 33, pp. 10718–10728.

Eric Wong and Zico Kolter (2018). “Provable defenses against adversarial examples via the convex outer adversarial polytope”. In: *International Conference on Machine Learning*.

Wei Xiao, Tsun-Hsuan Wang, Ramin Hasani, Mathias Lechner, Yutong Ban, Chuang Gan, and Daniela Rus (2023). “On the forward invariance of neural odes”. In: *International Conference on Machine Learning*, pp. 38100–38124.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh (2020). “Automatic Perturbation Analysis for Scalable Certified Robustness and beyond”. In: *Advances in Neural Information Processing Systems*.

Hanshu Yan, Jiawei Du, Vincent Y.F. Tan, and Jiashi Feng (2020). “On Robustness of Neural Ordinary Differential Equations”. In: *International Conference on Learning Representations*.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel (2018). “Efficient Neural Network Robustness Certification with General Activation Functions”. In: *Advances in Neural Information Processing Systems* 31.

Kaiqing Zhang, Bin Hu, and Tamer Basar (2020). “Policy Optimization for $\mathcal{H}_2$ Linear Control with $\mathcal{H}_\infty$ Robustness Guarantee: Implicit Regularization and Global Convergence”. In: *Learning for Dynamics and Control*, pp. 179–190.

Kemin Zhou and John Comstock Doyle (1998). *Essentials of Robust Control*. Vol. 104. Prentice Hall Upper Saddle River, NJ.

Chapter 5

RULE-GUIDED DIFFUSION MODELS VIA STOCHASTIC CONTROL

Yujia Huang, Adishree Ghatare, Yuanzhe Liu, Ziniu Hu, Qinsheng Zhang, Chandramouli Sastry, Siddharth Gururani, Sageev Oore, and Yisong Yue (2024). "Symbolic Music Generation with Non-Differentiable Rule Guided Diffusion". In: *arXiv preprint arXiv:2402.14285*. URL: https://arxiv.org/abs/2402.14285.

5.1 Introduction

We are interested in developing methods for controllable symbolic music generation. There has been rapid progress in the development of modern generative models for symbolic music (Huang et al., 2018; Huang and Yang, 2020; Hsiao et al., 2021; Min et al., 2023). To facilitate interaction between human composers and these models, it is crucial for these models to adhere to specific musical rules, such as chord progression, during the composition process. Moreover, these rules can be quite nuanced (e.g., the difference between a major chord and minor chord is very small).

A common method to incorporate rules in generative models is to train with rule labels (Choi et al., 2020; Wu and Yang, 2023; Rütte et al., 2022). However, integrating multiple musical rules during the training phase poses a significant challenge. Continuously updating model parameters to accommodate each new rule is not only costly but also will soon become impractical for compositions that involve many rules. Hence, there is a growing need for a method to guide pre-trained generative models in generating samples that conform to specific rules in a more flexible, light-weight, or plug-and-play manner.

Diffusion models (Ho et al., 2020; Song et al., 2021b) have emerged as a powerful generative modeling approach in many domains including images (Dhariwal and Nichol, 2021), audio (Huang et al., 2023) and video (Ho et al., 2022). A key feature of diffusion models is that they allow for post-hoc guidance of pre-trained models. Recent works have demonstrated success in guiding diffusion models with differentiable losses in a plug-and-play manner (Chung et al., 2023; Song et al.,

2023). Starting from Gaussian noise, diffusion models generate samples from coarse to fine. The key idea of guidance is to update each intermediate step with the gradient of the loss. However, there are still two challenges to generate symbolic music with rule guidance: First, many rules (e.g., note density) are not differentiable. Second, they may be black box APIs that hinder backpropagation.

To this end, we propose Stochastic Control Guidance (SCG), a new algorithm that enables plug-and-play guidance in diffusion models for non-differentiable rules. Our algorithm is inspired by stochastic control, where we pose the problem of generating samples that follow rule guidance as optimal control within a stochastic dynamical system. We obtain the analytical form of optimal control via path integral control theory (Theodorou et al., 2010), and adapt it to an efficient implementation within diffusion models. Specifically, we generate multiple realizations at each sampling step, and select the one that best follows the target (Figure 5.1). This process only requires forward evaluation of rule functions, making it applicable to non-differentiable rules.

To develop a practical overall framework, we also introduce a latent diffusion architecture with a transformer backbone for symbolic music generation. This architecture is able to generate dynamic music performances at 10ms time resolution, which is a significant challenge for standard pixel space diffusion models.

Our framework demonstrates state-of-the-art performance in various music generation tasks, offering superior rule guidance over popular methods and enabling

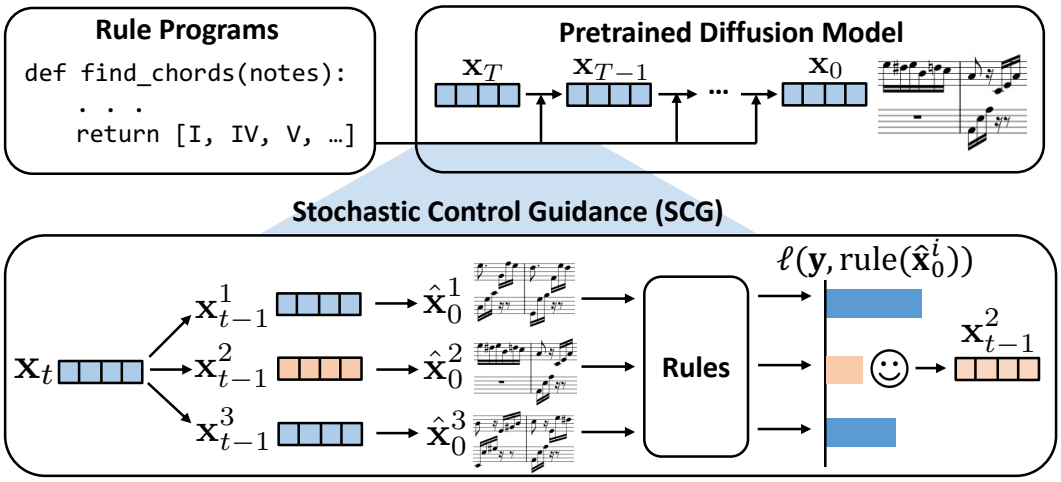


Figure 5.1: **Overview of Stochastic Control Guidance (SCG) for plug-and-play non-differentiable rule guided generation.** At each sampling step, we sample several realizations of the next step, and select the one yielding the most rule-compliant clean sample.

musicians to effectively use it as a compositional tool. Our code is available here.

In summary, our contributions are as follows:

- We introduce Stochastic Control Guidance (SCG), which achieves plug-and-play guidance in diffusion models for non-differentiable rules.
- We provide a theoretical justification of SCG from a stochastic control perspective.
- We introduce a latent diffusion model architecture for symbolic music generation with high time resolution.
- We demonstrate that our framework enables flexible, interpretable and controllable symbolic music generation in a variety of tasks.

5.2 Related Works

Current symbolic music generation methods are mainly divided into MIDI token-based and piano roll-based approaches. MIDI-based methods treat music as sequences of discrete tokens, often using transformers for MIDI token generation (Huang et al., 2018; Huang and Yang, 2020; Ren et al., 2020; Hsiao et al., 2021). Piano roll representations, resembling image formats with time on the horizontal axis and pitches vertically, have inspired the use of image generative models like GANs (Yang et al., 2017; Dong et al., 2018) for their generation. Recent efforts (Atassi, 2023; Min et al., 2023) apply diffusion models to generate binary, quantized piano rolls. Our work extends this by incorporating velocity and pedal information into piano rolls and employing a finer time resolution of 10 ms, thereby facilitating the generation of more dynamic piano performances.

Another line of research seeks to enhance control over certain attributes in the generated music. Some studies (Brunner et al., 2018; Roberts et al., 2018) have leveraged VAE models to learn a disentangled latent space, achieving controllability over specific attributes by manipulating latents in designated directions. Further, various works have conditioned LSTMs (Meade et al., 2019) or transformers on different factors like style (Choi et al., 2020), note density (Wu and Yang, 2023), or attributes like time signature, instruments, and chords (Rütte et al., 2022). However, these methods are limited to predefined attributes and are not easily extendable to new attributes due to the necessity of conditioning on labels during training.

Recent developments in the use of diffusion models for symbolic music generation have adapted controllable image generation techniques. Examples include generating

complementary parts given melody/accompaniment (inpainting), bridging two music segments (infilling) (Min et al., 2023), extending existing music pieces (outpainting), and generating piano rolls from stroke piano rolls (Zhang et al., 2023a). Yet, when it comes to rule-based guidance, existing approaches still require training on specific attributes, such as chord progression (Min et al., 2023; Li and Sung, 2023), limiting their adaptability for composers desiring to incorporate new rules. Our work enables flexible rule-based guidance via SCG. Additionally, our method is compatible with other diffusion model techniques like inpainting, outpainting, and editing, further enhancing its versatility in music generation.

The conceptualization of stochastic optimal control in diffusion models has spurred theoretical advancements and practical applications. Zhang and Chen (2021) employed celebrated path integral theory to transform a simple Ornstein–Uhlenbeck process to a novel process whose target distribution matches given marginal distribution. Further extending this framework, Berner et al. (2022) and Vargas et al. (2023) established a novel link between stochastic optimal control problems and generative models, interconnected through stochastic differential equations.

5.3 Background

Score-based diffusion models. Diffusion models generate data by reversing a diffusion process. Let $p(\mathbf{x})$ be the unknown data distribution, the forward diffusion process $\{\mathbf{x}_t\}_{t\in[0,T]}$ diffuse $p(\mathbf{x})$ to a noise distribution that is easy to sample from (e.g., standard Gaussian distribution). Song et al. (2021b) models the forward diffusion process as the solution to an SDE:

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}, \tag{5.1}$$

where the initial condition $\mathbf{x}_0 := \mathbf{x} \sim p(\mathbf{x})$, $\mathbf{f} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is the drift coefficient, $g : \mathbb{R} \to \mathbb{R}$ is the diffusion coefficient and $\mathbf{w} \in \mathbb{R}^d$ is a standard Wiener process.

Let $p_t(\mathbf{x})$ denote the marginal distribution of $\mathbf{x}_t$. The diffusion and drift coefficient can be properly designed such that $p_T(\mathbf{x}_)\approx \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$. In this paper, we consider the VP-SDE (Song et al., 2021b), where $\mathbf{f}(\mathbf{x}, t) := -\frac{1}{2}\beta(t)\mathbf{x}$ and $g(t) := \sqrt{\beta(t)}$, where $\beta(t)$ is a noise schedule. DDPM (Ho et al., 2020) can be regarded as a discretization of VP-SDE.

Samples are generated using the reverse-time SDE:

$$\mathrm{d}\mathbf{x}_t = \left[\mathbf{f}(\mathbf{x}_t, t) - g(t)^2\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)\right]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}_t, \tag{5.2}$$

where $\mathbf{f}(\mathbf{x}_t, t) : \mathbb{R}^d \to \mathbb{R}$ is the drift coefficient, $g : \mathbb{R} \to \mathbb{R}$ is the diffusion coefficient, dt is an infinitesimal negative time step and $\bar{\mathbf{w}}_t$ is a standard reverse-time Wiener process. Sampling $\mathbf{x}_T \sim p_T(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and solving the above SDE from $t = T$ to $t = 0$ produces samples from the data distribution: $\mathbf{x}_0 \sim p_0(\mathbf{x}) = p(\mathbf{x})$.

Since the data distribution is unknown, it is popular to approximate the score function $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ via a neural network $\mathbf{s}_\theta(\mathbf{x}, t)$ and train it with a weighted sum of denoising score matching objectives (Song et al., 2021b).

Classifier and Classifier-free Guidance. Guided diffusion models generates samples from $p(\mathbf{x}|\mathbf{y})$ given label $\mathbf{y}$. Classifier guidance (Dhariwal and Nichol, 2021) achieves this by training a classifier $p_t(\mathbf{y}|\mathbf{x}_t)$ on the noisy sample and label pair, and mix its gradient with the score of the diffusion model during sampling. The conditional score function becomes $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \omega \nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t)$, where ω is called guidance scale. This approximates the samples from the distribution $\tilde{p}(\mathbf{x}_t|\mathbf{y}) \propto p(\mathbf{x}_t)p(\mathbf{y}|\mathbf{x}_t)^\omega$. Classifier guidance is able to guide a pre-trained generative model at the cost of training an extra classifier on the noisy data.

Classifier-free guidance (Ho and Salimans, 2022) avoids training classifiers by jointly training conditional and unconditional diffusion models, and combining their score estimates during sampling. The mixed score function becomes $(1 + \omega)\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{y}) - \omega \log p(\mathbf{x}_t)$, where ω is the guidance strength. Despite easy implementation, it is expensive to extend classifier-free guidance to unknown or composite labels, because it requires re-training the diffusion model.

Loss-Guided Diffusion. To reduce the need of additional training for conditional generation, methods have been proposed to guided diffusion models to generate samples in a plug-and-play way. Instead of training a classifier to approximate $p(\mathbf{y}|\mathbf{x}_t)$, Diffusion Posterior Sampling (DPS) (Chung et al., 2023) uses $p(\mathbf{y}|\hat{\mathbf{x}}_0)$, where $\hat{\mathbf{x}}_0 := \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t]$ is obtained through the Tweedie's formula (Efron, 2011):

$$\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}(t)}}(\mathbf{x}_t + (1 - \bar{\alpha}(t))\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)). \tag{5.3}$$

Recall that $p(\mathbf{y}|\mathbf{x}_t)$ can be factorized as:

$$p(\mathbf{y}|\mathbf{x}_t) = \int p(\mathbf{y}|\mathbf{x}_0)p(\mathbf{x}_0|\mathbf{x}_t)\mathrm{d}\mathbf{x}_0 = \mathbb{E}_{\mathbf{x}_0 \sim p(\mathbf{x}_0|\mathbf{x}_t)} p(\mathbf{y}|\mathbf{x}_0).$$

DPS uses a point estimation of this quantity. Later work (Song et al., 2023) proposes to use Monte-Carlo estimation of this by sampling from approximated $p(\mathbf{x}_0|\mathbf{x}_t)$. However, these methods requires the loss function used to specify the condition to be differentiable. Many symbolic rules we consider in this paper are non-differentiable.

5.4 Non-Differentiable Rule Guidance

We now present Stochastic Control Guidance for non-differentiable rule guidance in diffusion models. We start with defining rule guidance in Section 5.4. Inspired by stochastic control (Section 5.4), we define a value function as a loss measuring (lack of) rule adherence, and show that optimal control steers the reverse diffusion to the target distribution. We then discuss practical algorithms (Section 5.4). We conclude by establishing a general theoretical connection that enables many guidance methods to be viewed through the lens of stochastic optimal control (Section 5.4).

Rule Guidance Problem

Assume that we have a pre-trained diffusion model that can sample from the data distribution $p(\mathbf{x})$, and a loss function $\ell_{\mathbf{y}} : \mathcal{X} \to \mathbb{R}$ that characterizes how well a sample follows some conditions $\mathbf{y}$: $p(\mathbf{y}|\mathbf{x}) \propto e^{-\ell_{\mathbf{y}}(\mathbf{x})}$. Our goal is to sample from the following distribution:

$$p(\mathbf{x}|\mathbf{y}) = p(\mathbf{x})\frac{e^{-\ell_{\mathbf{y}}(\mathbf{x})}}{Z} \propto p(\mathbf{x})p(\mathbf{y}|\mathbf{x}), \tag{5.4}$$

where $Z = \int_{\mathbf{x}} p(\mathbf{x})e^{-\ell_{\mathbf{y}}(\mathbf{x})}\mathrm{d}\mathbf{x}$.

A central challenge that we tackle is that many musical rules are non-differentiable, which makes sampling from Eq. 5.4 difficult. For instance, let $\mathbf{x} = [x_1, x_2, ..., x_n] \in [0, 1]^n$ be a vector where each x_i represents the volume of a note, so that the note density is computed as $\mathrm{ND}(\mathbf{x}) = \sum_{i=1}^{n} \mathbb{1}(x_i > \epsilon)$, where ϵ is a small number. Then the loss is defined as $\ell_y(\mathbf{x}) = |y - \mathrm{ND}(\mathbf{x})|$, which is non-differentiable.

Guidance via Stochastic Control

The pre-trained diffusion model generates samples using the reverse-time SDE (Eq. 5.2). Let $\boldsymbol{\eta}_t = \mathbf{x}_{T-t}$, and $\tilde{\mathbf{f}}(\boldsymbol{\eta}_t, t) = \mathbf{f}(\boldsymbol{\eta}_t, t) - g(t)^2\nabla_{\boldsymbol{\eta}_t} \log p_t(\boldsymbol{\eta}_t)$. We can rewrite Eq. 5.2 as:

$$\mathrm{d}\boldsymbol{\eta}_t = \tilde{\mathbf{f}}(\boldsymbol{\eta}_t, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}_t, \tag{5.5}$$

where $\mathrm{d}t$ is an infinitesimal time step and $\mathrm{d}\mathbf{w}$ is a standard Wiener process. Sampling $\boldsymbol{\eta}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and solving the above SDE from $t = 0$ to $t = T$ produces samples from the data distribution.

We want to find a control $\mathbf{u}(\boldsymbol{\eta}_t, t)$, such that solving the following SDE yields samples from target distribution $p(\boldsymbol{\eta}|\mathbf{y})$:

$$\mathrm{d}\boldsymbol{\eta}_t = \tilde{\mathbf{f}}(\boldsymbol{\eta}_t, t)\mathrm{d}t + g(t)(\mathbf{u}(\boldsymbol{\eta}_t, t)\mathrm{d}t + \mathrm{d}\mathbf{w}_t). \tag{5.6}$$

We use $\mathbf{u}_t := \mathbf{u}(\boldsymbol{\eta}_t, t)$ and $\tilde{\mathbf{f}}_t := \tilde{\mathbf{f}}(\boldsymbol{\eta}_t, t)$ for brevity, noting they are state-dependent.

Considering the stochastic dynamical system in Eq. 5.6 for $0 \leq t \leq T$ and initial state $\boldsymbol{\eta}_0 = \bar{\boldsymbol{\eta}}_0$, we address the optimal control problem associated with the cost function $C_u(\boldsymbol{\eta}_t, t)$, which is defined as the expectation over all stochastic trajectories starting at $\boldsymbol{\eta}_t$ with control function $\mathbf{u}_t$:

$$C_u(\boldsymbol{\eta}_t, t) = \mathbb{E}\left[\phi(\boldsymbol{\eta}_T) + \int_t^T \frac{1}{2}\|\mathbf{u}_t\|^2 \mathrm{d}t\right]. \tag{5.7}$$

It is known that the optimal control policy admits an analytical solution (Pavon, 1989):

$$\mathbf{u}_t^* = -g(t)\nabla_{\boldsymbol{\eta}} V(\boldsymbol{\eta}, t), \tag{5.8}$$

where function $V(\boldsymbol{\eta}, t)$, known as the *value* function, is the solution to celebrated stochastic Hamilton-Jacobi-Bellman (HJB) equation (Evans, 2022):

$$\begin{aligned} -\partial_t V(\boldsymbol{\eta}, t) = &-\frac{1}{2} g(t)^2 (\nabla_{\boldsymbol{\eta}} V)^\top (\nabla_{\boldsymbol{\eta}} V) \\ &+ (\nabla_{\boldsymbol{\eta}} V)^\top \tilde{\mathbf{f}}_t + \frac{1}{2} g(t)^2 \mathrm{Tr}(\nabla^2_{\boldsymbol{\eta}\boldsymbol{\eta}} V), \end{aligned} \tag{5.9}$$

with boundary condition $V(\boldsymbol{\eta}, T) = \phi(\boldsymbol{\eta})$.

Path Integral Control. Although solving HJB in Eq. 5.9 is nontrivial due to its non-linearity w.r.t. V, using an exponential transformation $\Psi(\boldsymbol{\eta}, t) = e^{-V(\boldsymbol{\eta}, t)}$ yields a linear HJB equation in Ψ:

$$-\partial_t \Psi(\boldsymbol{\eta}, t) = \left(\tilde{\mathbf{f}}_t^\top \nabla_{\boldsymbol{\eta}} + \frac{1}{2} g(t)^2 \mathrm{Tr}(\nabla^2_{\boldsymbol{\eta}\boldsymbol{\eta}})\right) \Psi(\boldsymbol{\eta}, t), \tag{5.10}$$

with boundary condition $\Psi(\boldsymbol{\eta}, T) = e^{-\phi(\boldsymbol{\eta})}$. We call Ψ the *desirability* function as it is inversely related to the value V.

Let $\Omega = C([0, T]; \mathbb{R}^d)$ be the space consisting of all possible continuous-time stochastic trajectories $\tau = \{\boldsymbol{\eta}_t, 0 \leq t \leq T\}$, and $\mathcal{Q}^0$ be the measure induced by an uncontrolled stochastic process (Eq. 5.5). Then the linear HJB equation has the following solution according to the Feynman-Kac formula (Øksendal, 2003):

$$\Psi(\boldsymbol{\eta}, t) = \mathbb{E}_{\mathcal{Q}^0}\left[e^{-\phi(\boldsymbol{\eta}_T)} | \boldsymbol{\eta}_t = \boldsymbol{\eta}\right]. \tag{5.11}$$

Eq. 5.11 shows that the value function can be computed by *only forward sampling the uncontrolled process* without knowing the optimal control policy. Plugging Eq 5.11 into Eq 5.8 yields the analytic optimal policy, which aligns with the well-known path

integral control approach (Theodorou et al., 2010; Theodorou, 2015; Fleming and Mitter, 1982):

$$\mathbf{u}_t^*(\boldsymbol{\eta})\mathrm{d}t = g(t)\nabla_{\boldsymbol{\eta}} \log \Psi(\boldsymbol{\eta}, t)\mathrm{d}t \tag{5.12}$$

$$= \frac{\mathbb{E}_{\mathcal{Q}^0}\left[e^{-\phi(\boldsymbol{\eta}_T)}\mathrm{d}\mathbf{w}_t | \boldsymbol{\eta}_t = \boldsymbol{\eta}\right]}{\mathbb{E}_{\mathcal{Q}^0}\left[e^{-\phi(\boldsymbol{\eta}_T)} | \boldsymbol{\eta}_t = \boldsymbol{\eta}\right]}. \tag{5.13}$$

Next, we show that using the above optimal control, we can guide the generation process to produce samples from the target conditional distribution $p(\boldsymbol{\eta}|\mathbf{y})$.

Theorem 5.4.1 (proof in Appendix D.1)**.** *Consider the dynamical system in Eq. 5.6. For a terminal cost defined as* $\phi(\boldsymbol{\eta}_T) \triangleq \ell_y(\boldsymbol{\eta}_T) \triangleq -\log p(\mathbf{y}|\boldsymbol{\eta}_T) + \texttt{const}$*, and initial condition* $\boldsymbol{\eta}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$*, the terminal distribution induced by the optimal control policy* $\mathbf{u}_t^*$ *(Eq. 5.13) is:*

$$\mathcal{Q}^*(\boldsymbol{\eta}_T) = p(\boldsymbol{\eta}_T|\mathbf{y}). \tag{5.14}$$

Algorithm 4 Stochastic Control Guided DDPM sampling

Require: Loss function ℓ_y, rule target $\mathbf{y}$, number of samples n.
  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  for $t = T$ **to** 1 **do**
    ▷ Compute the posterior mean of $\mathbf{x}_{t-1}$.
    $\hat{\mathbf{x}}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta\left(\mathbf{x}_t, t\right)\right)$
    if $t > 1$ **then**
      ▷ Sampling possible next steps.
      $\mathbf{x}_{t-1}^i = \hat{\mathbf{x}}_{t-1} + \sigma_t \mathbf{z}^i$, with $\mathbf{z}^1, ..., \mathbf{z}^n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
      ▷ Estimate the clean sample from noisy sample.
      $\hat{\mathbf{x}}_0^i = \frac{1}{\sqrt{\bar{\alpha}_{t-1}}}\left(\mathbf{x}_{t-1}^i - \sqrt{1-\bar{\alpha}_{t-1}}\epsilon_\theta\left(\mathbf{x}_{t-1}^i, t-1\right)\right)$
      ▷ Find the direction that minimizes the loss.
      $k = \arg\max_i \log p(y|\hat{\mathbf{x}}_0^i) = \arg\max_i -\ell_y(\hat{\mathbf{x}}_0^i)$
      $\mathbf{x}_{t-1} = \mathbf{x}_{t-1}^k$
    else
      $\mathbf{x}_{t-1} = \hat{\mathbf{x}}_{t-1}$
    end if
  end for
  return: $\mathbf{x}_0$

Practical Algorithms

Approximation of the Optimal Control. In practice, it is expensive to compute Eq. 5.13, because one needs to unroll the whole trajectory to get $\boldsymbol{\eta}_T$. Instead of using

Eq. 5.13 as our optimal control, we set $\mathbf{u}_t \mathrm{d}t + \mathrm{d}\mathbf{w}$ to the following:

$$\arg\max_{\mathrm{d}\mathbf{w}_t} -\ell_y(\hat{\boldsymbol{\eta}}_T), \tag{5.15}$$

where $\hat{\boldsymbol{\eta}}_T = \mathbb{E}\left[\boldsymbol{\eta}_T | \boldsymbol{\eta}_{t+\mathrm{d}t}\right]$ can be obtained via Tweedie's Formula (Eq. 5.3), which is a one-step computation and much cheaper than solving the whole trajectory.

Eq. 5.15 is an approximation to a tempered version of Eq. 5.13. Consider the terminal cost is defined with a scaling factor K, i.e., $\phi(\boldsymbol{\eta}_T) = \ell_y(\boldsymbol{\eta}_T)/K$. When $K \to 0$, Eq. 5.13 becomes:

$$\arg\max_{\mathrm{d}\mathbf{w}_t} \max_{\tau} -\ell_y(\boldsymbol{\eta}_T | \boldsymbol{\eta}_{t+\mathrm{d}t}), \tag{5.16}$$

where $\boldsymbol{\eta}_{t+\mathrm{d}t} = \boldsymbol{\eta}_t + \tilde{\mathbf{f}}(\boldsymbol{\eta}_t, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}_t$, and $\tau : [t + \mathrm{d}t, T] \to \mathbb{R}^d$ represents a trajectory. The solution of Eq. 5.15 optimizes a lower bound of the objective in Eq. 5.16:

$$\max_{\mathrm{d}\mathbf{w}_t, \tau} -\ell_y(\boldsymbol{\eta}_T | \boldsymbol{\eta}_{t+\mathrm{d}t}) \geq \max_{\mathrm{d}\mathbf{w}_t} -\ell_y(\mathbb{E}\left[\boldsymbol{\eta}_T | \boldsymbol{\eta}_{t+\mathrm{d}t}\right]). \tag{5.17}$$

Intuition. Our SCG algorithm implemented with DDPM sampling (Ho et al., 2020) is outlined in Algorithm 4 and illustrated in Figure 5.1, where we use $\mathbf{x}_t \triangleq \boldsymbol{\eta}_{T-t}$ to denote the intermediate states following conventions of diffusion model notations. The intuition is that we select the direction that leads to the most probable sample at each step. For every step t in the sampling process, given $\mathbf{x}_t$, we compute multiple realizations of the next step $\mathbf{x}_{t-1}$, estimate the corresponding clean sample $\hat{\mathbf{x}}_0$, and choose the $\mathbf{x}_{t-1}$ that leads to the lowest loss $\ell_{\mathbf{y}}(\hat{\mathbf{x}}_0)$. Notably, we only need to evaluate the forward pass of the rule function, and there is no need to evaluate or estimate its gradient, making our method suitable for non-differentiable and black-box rule functions. Furthermore, it is also compatible with other stochastic sampling procedure in diffusion models (Appendix D.2).

General Theoretical Connection

In this section, we show a general connection (Proposition 5.4.1) that enables many guidance methods to be viewed through the lens of stochastic optimal control.

Proposition 5.4.1 (proof in Appendix D.1)**.** *Consider the dynamical system in Eq. 5.6 with terminal cost* $\phi(\boldsymbol{\eta}_T) \triangleq -\log p(\mathbf{y}|\boldsymbol{\eta}_T) + \texttt{const}$. *We have:* $\Psi(\boldsymbol{\eta}_t, t) = c \cdot p(\mathbf{y}|\boldsymbol{\eta}_t)$.

Proposition 5.4.1 says that the desirability function equals to the likelihood function. Then many popular guidance techniques can be seen as different implementations of

the optimal control following Eq. 5.12):

$$g(t)\nabla_{\boldsymbol{\eta}_t} \log p(\mathbf{y}|\boldsymbol{\eta}_t) = g(t)\nabla_{\boldsymbol{\eta}_t} \log \Psi(\boldsymbol{\eta}_t, t) = \mathbf{u}_t^*(\boldsymbol{\eta}_t).$$

Classifier guidance (Dhariwal and Nichol, 2021) trains a neural network on noisy data pair $\{\boldsymbol{\eta}_t, \mathbf{y}\}$ to approximate $\Psi(\boldsymbol{\eta}_t, t)$, and differentiate through it to obtain $\mathbf{u}_t^*(\boldsymbol{\eta})$.

DPS (Chung et al., 2023) avoids training a surrogate model by approximating $\Psi(\boldsymbol{\eta}_t, t)$ with $\Psi(\hat{\boldsymbol{\eta}}_T, T)$, where $\hat{\boldsymbol{\eta}}_T$ is the posterior mean that can be obtained through the Tweedie's formula (Eq. 5.3). Since $\nabla_{\boldsymbol{\eta}_t}\Psi(\hat{\boldsymbol{\eta}}_T, T) = \frac{\partial \Psi(\hat{\boldsymbol{\eta}}_T, T)}{\partial \hat{\boldsymbol{\eta}}_T}\frac{\partial \hat{\boldsymbol{\eta}}_T}{\partial \boldsymbol{\eta}_t}$, it requires $\Psi(\hat{\boldsymbol{\eta}}_T, T) \propto e^{-\ell_{\mathbf{y}}(\hat{\boldsymbol{\eta}}_T)}$ to be differentiable.

In contrast, our approach is inspired by path integral control, and only needs the forward evaluation of the rule function (Eq. 5.15). Therefore, our method does not require the rule function to be differentiable.

5.5 Latent Diffusion Architecture

To arrive at a practical overall framework, we develop a latent diffusion architecture tailored towards symbolic music generation, and in particular able to generate at 10ms time resolution. This architecture can be combined with Stochastic Control Guidance in a plug-and-play fashion.

Data Representation. We represent symbolic music as a 3-channel tensor. Each column in this representation accounts for a 10 ms timeframe. The first channel is the piano roll, where horizontal axis represents time and vertical axis represents pitch. Each element takes value from 0-127, indicating the velocity (volume) of the note. The second channel is the onset roll, consisting of binary values that denote the presence of note onsets. The third channel is the pedal roll, representing the sustain pedal control for each timeframe.

Model architecture. We first use a VAE model to encode short segments of piano rolls of shape $3 \times 128 \times 128$ into a latent space. Then we concatenate the latent codes and train a diffusion model to capture their joint distribution (Figure 5.2). For the VAE, we use the U-Net backbone following (Rombach et al., 2022). The training involves a denoising objective in conjunction with KL regularization: we introduce musically semantic perturbations (such as adding adjacent notes) to the data and train the model to revert to the original, unperturbed data. Both KL regularization and the denoising objective have proven indispensable for developing diffusion models with robust generative capabilities in subsequent stages.

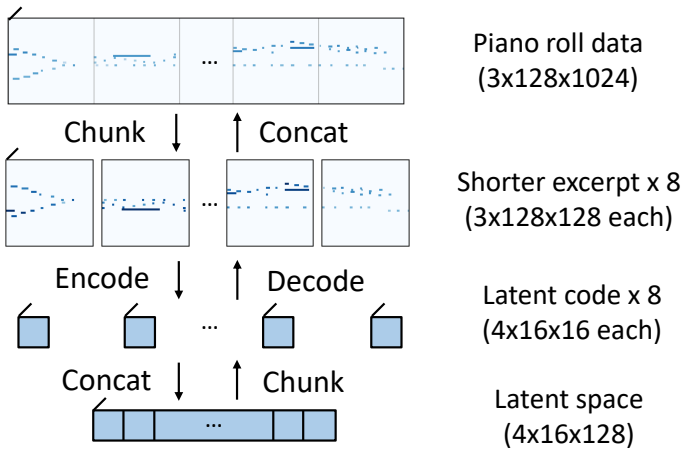


Figure 5.2: **We use a VAE to encode piano roll segments to latent space and concatenate them for the next stage of diffusion training.**

For the diffusion model, we use the DiT architecture (Peebles and Xie, 2023). In contrast to the standard U-Net, the transformer backbone is more adept at handling sequences of latent tokens. Rather than absolute position encoding, we use rotary position embedding (Su et al., 2023) to better generalize across various input lengths. We train the diffusion model on piano rolls of length 1024 (10.24 s). To generate musical excerpts of arbitrary length, we apply DiffCollage (Zhang et al., 2023b) to aggregate the score function of shorter music segments.

5.6 Experiments

We evaluate our method on a wide range of symbolic music generation tasks: unconditional generation (Sec 5.6), individual rule guidance (Sec 5.6), composite rule guidance (Sec 5.6) and editing (Appendix D.3). We perform ablation studies in Sec 5.6 and subjective evaluation in Sec 5.6. In addition, we demonstrate that our method can be used as a compositional tool for musicians in Sec 5.6.

Experimental Settings

Data. We train our model on several piano midi datasets that cover both classical and pop genres. The MAESTRO dataset (Hawthorne et al., 2019) has about 1200 pieces of classical piano performances with expressive dynamics, resulting in about 200 hours of music performance. In addition, we crawled about 14k MIDI files from Muscore in classical, religion and soundtrack genres across all skill levels, yielding about 700 hours of data. We also used two Pop piano datasets: Pop1k7 (Hsiao et al., 2021) and Pop909 (Wang et al., 2020) that contain 108 hours and 60 hours of pop

piano midi translated from audio respectively.

Training and Inference Setup. We first train a VAE model to encode piano rolls to latent space, then fix the VAE and train a diffusion model on this space. The diffusion model is trained with dataset-based conditioning: classical performance (Maestro), classical sheet music (Muscore) and Pop (pop1k7 and pop909), following Classifier-free Guidance (Ho and Salimans, 2022) with a dropout rate of 0.1. We train the model for 1.2M steps and use DDPM (Ho et al., 2020) with 1000 steps as the default sampling method unless stated otherwise. All experiments are run on NVIDIA A100-SXM4 GPUs.

Unconditional Generation

Baselines. We compare with state-of-the-art symbolic music generators trained on various datasets (Table 5.1).

Method	Model	Dataset	Representation
MusicTr (Huang et al., 2018)	Transformer	Maestro	MIDI-like
Remi (Huang and Yang, 2020)	Transformer	Pop775	REMI
CPW (Hsiao et al., 2021)	Transformer	Pop1k7	CP
PolyDiff (Min et al., 2023)	Diffusion	POP909	Piano roll

Table 5.1: Baselines for unconditional music generation.

Objective Metrics. It is worth mentioning that quantitative evaluation of music quality remains an open problem (Yin et al., 2023). Nevertheless, we use the average overlapping area (OA) between the intra-set and inter-set distribution of 7 musical attributes (pitch range, note density, etc.) proposed in (Yang and Lerch, 2020) as the objective metric for music quality. As a sanity check, we compare a subset of the training dataset with another subset (denoted by GT in Table 5.2), and find that GT on all the datasets achieves the highest average OA. This indicates that this metric is a reasonable necessary condition for good generated music quality.

Results. The evaluation results are in Table 5.2, highlighting the highest values (excluding GT) in bold. Our method achieves the highest average OA on all the datasets. The baselines are trained on individual dataset, and do not generalize well across datasets. MusicTr has the second-best overall rating for classical music (Maestro and Muscore), while it holds the lowest rating for pop music. CPW, on the other hand, ranks second in pop music but has the lowest rating in classical music. In contrast, our model delivers strong performance consistently across all the datasets.

Dataset	GT	MusicTr	Remi	CPW	PolyDiff	**Ours**
Maestro	0.944 ± 0.002	0.903 ± 0.005	0.847 ± 0.005	0.801 ± 0.006	0.842 ± 0.007	$\mathbf{0.943 \pm 0.003}$
Muscore	0.945 ± 0.004	0.901 ± 0.004	0.879 ± 0.006	0.843 ± 0.007	0.845 ± 0.004	$\mathbf{0.934 \pm 0.003}$
Pop	0.957 ± 0.002	0.845 ± 0.004	0.866 ± 0.004	0.899 ± 0.005	0.883 ± 0.004	$\mathbf{0.939 \pm 0.004}$

Table 5.2: Average Overlapping Area (OA) across seven music attributes for unconditional generation, with highest non-GT OA bolded.

Individual Rule Guidance

Setup. We consider three rules: pitch histogram, note density (vertical and horizontal) and chord progression, where pitch histogram is differentiable and the other two are non-differentiable (see Appendix D.4 for the full definition of each rule). In our evaluation of the guidance performance, we default to conditioning on the Muscore dataset unless otherwise specified, owing to its comprehensive variety and extensive coverage of a broad spectrum of rule labels. For each rule, we randomly select 200 samples from the test dataset, and extract their attributes as the target for guided generation. We choose the number of samples to be 16 for SCG if without explicit mentioning.

Baselines. We compare with two popular post-hoc guidance methods: classifier guidance (Dhariwal and Nichol, 2021) and Diffusion Posterior Sampling (DPS) (Chung et al., 2023). For classifier guidance, we train a classifier on noisy latent and target pair for each rule. DPS only requires the loss to be defined on clean data $\mathbf{x}_0$ so we can directly plug in the rule in the loss if the rule is differentiable (DPS-Rule) without any additional training. However, it still requires the gradient of the rule, and therefore we train a surrogate model (a neural network) for non-differentiable rules (DPS-NN).

Results. Table 5.3 shows the loss between the generated attributes and the target attributes, which measures the adherence to the rule. We make three main observations. First, our method significantly outperforms the other methods on *non-differentiable* rules (note density and chord progression). It achieves the lowest loss, without need for training any surrogate model, which is mandatory for classifier guidance and DPS-NN.

Second, we find it challenging to train neural network surrogate models to approximate non-differentiable rules (Appendix D.5), leading to poor performance of guidance methods that rely on surrogate models. For differentiable rules (pitch histogram), the surrogate model learns well and DPS-NN achieves the lowest loss.

Third, to the best of our knowledge, our method is the first plug-and-play guidance

Method	Pitch Histogram ↓	Note Density ↓	Chord Progression ↓
No Guidance	0.018 ± 0.010	2.486 ± 3.530	0.831 ± 0.142
Classifier	0.005 ± 0.004	0.698 ± 0.587	0.723 ± 0.200
DPS - NN	$\mathbf{0.001 \pm 0.002}$	1.261 ± 2.340	0.414 ± 0.256
DPS - Rule	0.010 ± 0.008	2.508 ± 2.798	-
SCG (ours)	0.003 ± 0.004	$\mathbf{0.131 \pm 0.325}$	$\mathbf{0.2725 \pm 0.1637}$

Table 5.3: Loss between the target and the generated attributes for individual rule guidance. SCG significantly improves the controllability of non-differentiable rules.

method that supports non-differentiable and black-box loss functions. In contrast, DPS-rule fails to guide on note density because the gradient is zero almost everywhere. It also does not apply to chord progression because the loss involves a black-box API that cannot be back-propagated through. Overall, our method proves especially beneficial for guiding the generation process with non-differentiable loss functions, or for achieving guidance without the need for additional training.

Composite Rule Guidance

Method	Pitch Histogram ↓	Note Density ↓	Chord Progression ↓
No Guidance	0.018 ± 0.010	2.486 ± 3.530	0.831 ± 0.142
Classifier	0.006 ± 0.006	0.822 ± 0.844	0.724 ± 0.205
DPS-NN	0.004 ± 0.006	1.366 ± 2.265	0.661 ± 0.257
SCG	0.014 ± 0.009	0.466 ± 0.648	0.446 ± 0.205
SCG + DPS-NN	$\mathbf{0.002 \pm 0.007}$	0.238 ± 0.531	0.313 ± 0.231
SCG + Classifier	0.003 ± 0.005	$\mathbf{0.148 \pm 0.203}$	$\mathbf{0.284 \pm 0.197}$

Table 5.4: Loss between the target and the generated attributes for individual rule guidance. SCG + Classifier achieves significantly lower losses for all three rules simultaneously.

We apply our method to generate samples that follow composite rules, following the same setup in Section 5.6, and assuming that the rule labels are conditionally independent given the sample. For classifier and DPS-NN, we train a surrogate model for each rule, and combine the gradient of each classifier to obtain the guidance term: $\Sigma_i \omega_i \nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}_i|\mathbf{x}_t)$. For our method, we use a weighted loss function $\Sigma_i \omega_i \ell_{\mathbf{y}_i}(\mathbf{x})$ to select the best direction for each step. We set the weights on pitch histogram, note density and chord progression to be 40, 1, 1 respectively so that their loss is on the same order of magnitude.

The results are presented in Table 5.4. Our method achieves much lower loss on non-differentiable rules compared to other methods, similar to the case of individual rule guidance. However, it compromises control over the pitch histogram. Additionally,

the loss associated with each rule is higher than in scenarios of individual rule guidance, which aligns with expectations. This increase in loss occurs because it is more challenging to identify a direction that satisfies multiple rules simultaneously, as opposed to a single rule, within the same computational budget.

To enhance the controllability of composite rules, we integrate our SCG approach with gradient-based guidance methods. In this framework, the gradient of the surrogate model provides a preliminary guidance signal. SCG then identifies the optimal directions along these initially guided trajectories. As indicated in Table 5.4, this combination of our method with the baseline gradient method results in improved controllability for each rule, compared to the baseline method alone. Furthermore, we achieve a level of controllability comparable to that of individual rule guidance, while using the same number of samples.

Ablation Studies

Controllability and Computational Time Trade-off. Table 5.5 shows the loss achieved by SCG with different number of samples at each step. The time is reported for generating 4 samples in a batch. As anticipated, more samples results in lower loss, but requires more time. To achieve a balance between controllability and computational efficiency, we integrate classifier guidance with SCG. This combination yields interesting results: number of samples of 4, when used in conjunction with classifier guidance, delivers similar performance to number of samples of 16 with SCG alone, but is approximately four times faster.

Controllability and Quality Trade-Off. We observe a trade-off between controllability (measured by loss) and quality (measured by OA) in Table 5.5, where we guide the model to generate music following given note density. This is because there are more constraints on the generated music. For instance, a generative model could generate music that follows the note density exactly, but completely ignore the pitch of the notes. SCG achieves significant lower loss than the other guidance methods while maintaining reasonably high music quality. In addition, one can tune the balance between controllability and quality by tuning the number of samples at each step.

Impact of Sampling Strategy. By default, we use DDPM with 1000 steps as the base sampling algorithm, and apply SCG for rule guidance after 250 steps ($t = 750$). The reason that we do not start SCG from the beginning is that the decoded piano rolls at the beginning are quite sparse after thresholding the background. Consequently,

Method	n	Loss $\downarrow$	OA $\uparrow$	Time (s)
Classifier	1	0.698 ± 0.587	0.914 ± 0.006	47.8
DPS-NN	1	1.261 ± 2.340	0.735 ± 0.012	109.3
SCG	4	0.318 ± 0.770	$\mathbf{0.895 \pm 0.006}$	277.7
	8	0.214 ± 0.368	0.877 ± 0.006	531.6
	16	$\mathbf{0.131 \pm 0.325}$	0.880 ± 0.003	1242.6
Classifier + SCG	4	0.151 ± 0.298	$\mathbf{0.906 \pm 0.006}$	301.9
	8	0.098 ± 0.179	0.893 ± 0.004	555.6
	16	$\mathbf{0.064 \pm 0.159}$	0.899 ± 0.007	1253.9

Table 5.5: Trade-offs between controllability, quality and computational time. n refers to number of samples at each step.

the losses between the generated attributes and target attributes are almost the same among different realizations at this stage, making it ineffective for selecting the best directions.

To reduce the computational cost, we explore various sampling strategies, as detailed in Table 5.6. Firstly, we experimented with applying SCG intermittently, every k steps ($k = 2, 5$), and specifically during either the initial phase (750-400) or the latter phase (400-0) of the DDPM-1000 process. Among these variants, conducting SCG every 2 steps yielded the lowest loss. While the loss remains higher than in our default setting, this approach is about twice as fast. Additionally, we observed that applying SCG during the early phase of the process is more effective than in the later phase, likely due to greater perturbations early on, which enhance the likelihood of identifying optimal directions (see Appendix D.6 for more details).

Secondly, we considered early stopping of the DDPM-1000 process after k steps ($k = 800, 700$). This is motivated by our use of post-processing techniques like thresholding and smoothing note velocity on piano rolls, which reduces the need for fine-tuning in the latter stages of the generation process. Early stopping at 200 steps resulted in only marginally inferior outcomes but cut computational time by a quarter.

Finally, we explore the compatibility of SCG with other popular sampling algorithm for diffusion models, such as DDIM (Song et al., 2021a). By default, DDIM is deterministic. However, our SCG algorithm needs stochasticity to search for the best direction. Therefore, we set stochasticity $\eta = 1$ in the DDIM algorithm and refer the modified algorithm as stochastic DDIM (sDDIM). We tested sDDIM with 100, 50 and 25 steps. More steps offers lower loss and better music quality at a cost of longer sampling time.

Method	Guided Steps	Loss ↓	OA ↑	Time (s)
DDPM-1000	750-0	$\mathbf{0.131 \pm 0.325}$	0.880 ± 0.003	1242.6
	every 2	0.365 ± 0.559	0.893 ± 0.006	635.4
	every 5	0.632 ± 0.577	0.879 ± 0.005	269.8
	750-400	0.458 ± 0.647	0.902 ± 0.009	594.7
	400-0	1.297 ± 1.772	$\mathbf{0.912 \pm 0.007}$	674.6
DDPM†-800	750-200	$\mathbf{0.183 \pm 0.341}$	$\mathbf{0.864 \pm 0.005}$	912.6
DDPM†-700	750-300	1.950 ± 1.344	0.737 ± 0.011	747.3
sDDIM-100	all	$\mathbf{0.303 \pm 0.509}$	$\mathbf{0.887 \pm 0.005}$	164.3
sDDIM-50	all	0.372 ± 0.915	0.879 ± 0.008	81.9
sDDIM-25	all	0.428 ± 0.683	0.859 ± 0.005	40.7

Table 5.6: Impact of sampling strategy. The numbers that follow the method names are the total sampling steps. †: early stopping.

Subjective Evaluation

To compare performance of our SCG algorithm and baselines (classifier guidance and DPS), we carried out a listening test. We crafted four sets of rules (each set comprised of PH, ND, and CP), and use each method to generate samples that follow the rules, yielding a total of 12 samples, each 10.24 seconds long. Experienced listeners assess the quality of samples in 4 dimensions: rule alignment, musical creativity, musical coherence, and overall rating. In figure 5.3, SCG consistently outperforms the baselines in all dimensions. For details of our survey, please see Appendix D.8.

Examples of Our System as a Compositional Tool

To demonstrate how our system can be used effectively as a compositional tool, we provide links to three example videos, available through this website. For each video, a musician first indicated desired musical characteristics in terms of the rules (e.g. fairly sparse excerpt, following a simple I-V chord progression in C major, etc). The musician's plan was to then loop this and use that as an accompaniment over

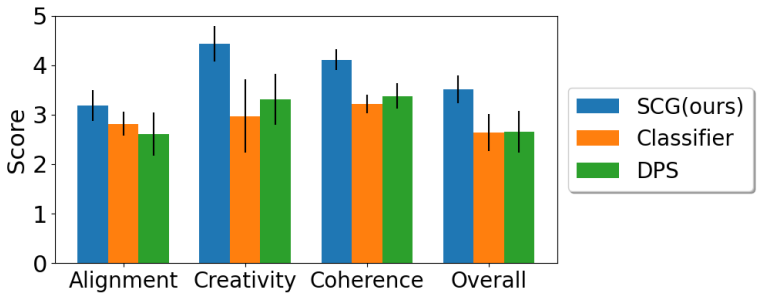


Figure 5.3: Subjective evaluation scores.

which they would then improvise. The system generated 3-5 options (i.e. samples) for each such request, and the musician chose their preferred sample with which to work, from each set. They sent the generated MIDI output to a Disklavier piano and then recorded a second track over top of that. In the accompanying videos (linked below), you can see what the musician is playing, and everything "else" that you hear is from the material that was generated by the system.

In **Video 1**, the model generated material that suggested a melody. Since the musician wanted to play the second track in the upper register, they first allowed the excerpt to play in full, as generated, and then removed the upper notes from the accompaniment to give room for themselves to play overtop. They chose to use the model's generated melody as a motif, and further improvise based on it.

In **Video 2**, the model generates an excerpt with a steady accompanying triplet ostinato behind a slower-moving descending melody in C minor (that suggests a progression that moves between the I and the V).

In **Video 3**, the model generates a sample with a changing note density and texture, and a slightly ambiguous harmonic quality that allowed flexibility in the improvising over it.

References

Lilac Atassi (2023). "Generating Symbolic Music using Diffusion Models". In: *arXiv preprint:2303.08385*.

Julius Berner, Lorenz Richter, and Karen Ullrich (2022). "An Optimal Control Perspective on Diffusion-based Generative Modeling". In: *arXiv preprint arXiv:2211.01364*.

Gino Brunner, Andres Konrad, Yuyi Wang, and Roger Wattenhofer (2018). "MIDI-VAE: Modeling Dynamics and Instrumentation of Music with Applications to Style Transfer". In: *19th International Society for Music Information Retrieval Conference*.

Kristy Choi, Curtis Hawthorne, Ian Simon, Monica Dinculescu, and Jesse Engel (2020). "Encoding Musical Style with Transformer Autoencoders". In: *International Conference on Machine Learning*, pp. 1899–1908.

Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye (2023). "Diffusion Posterior Sampling for General Noisy Inverse Problems". In: *International Conference on Learning Representations*.

Prafulla Dhariwal and Alexander Nichol (2021). "Diffusion Models Beat GANs on Image Synthesis". In: *Advances in Neural Information Processing Systems* 34, pp. 8780–8794.

Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang (2018). "MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Bradley Efron (2011). "Tweedie's Formula and Selection Bias". In: *Journal of the American Statistical Association* 106.496, pp. 1602–1614.

Lawrence C Evans (2022). *Partial Differential Equations*. Vol. 19. American Mathematical Society.

Wendell H Fleming and Sanjoy K Mitter (1982). "Optimal Control and Nonlinear Filtering for Nondegenerate Diffusion Processes". In: *Stochastics: An International Journal of Probability and Stochastic Processes* 8.1, pp. 63–77.

Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck (2019). "Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset". In: *International Conference on Learning Representations*.

Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. (2022). "Imagen Video: High Definition Video Generation with Diffusion Models". In: *arXiv preprint arXiv:2210.02303*.

Jonathan Ho, Ajay Jain, and Pieter Abbeel (2020). "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems* 33, pp. 6840–6851.

Jonathan Ho and Tim Salimans (2022). "Classifier-free Diffusion Guidance". In: *arXiv preprint arXiv:2207.12598*.

Wen-Yi Hsiao, Jen-Yu Liu, Yin-Cheng Yeh, and Yi-Hsuan Yang (2021). "Compound Word Transformer: Learning to Compose Full-song Music Over Dynamic Directed Hypergraphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 1, pp. 178–186.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck (2018). "Music Transformer". In: *arXiv preprint arXiv:1809.04281*.

Qingqing Huang, Daniel S Park, Tao Wang, Timo I Denk, Andy Ly, Nanxin Chen, Zhengdong Zhang, Zhishuai Zhang, Jiahui Yu, Christian Frank, et al. (2023). "Noise2music: Text-conditioned Music Generation with Diffusion Models". In: *arXiv preprint arXiv:2302.03917*.

Yu-Siang Huang and Yi-Hsuan Yang (2020). "Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions". In: *Proceedings of the 28th ACM international conference on multimedia*, pp. 1180–1188.

Shuyu Li and Yunsick Sung (2023). “MelodyDiffusion: Chord-Conditioned Melody Generation Using a Transformer-Based Diffusion Model”. In: *Mathematics* 11.8, p. 1915.

Nicholas Meade, Nicholas Barreyre, Scott C Lowe, and Sageev Oore (2019). “Exploring Conditioning for Generative Music Systems with Human-interpretable Controls”. In: *arXiv preprint arXiv:1907.04352*.

Lejun Min, Junyan Jiang, Gus Xia, and Jingwei Zhao (2023). “Polyffusion: A Diffusion Model for Polyphonic Score Generation with Internal and External Controls”. In: *Proc. of the 24th Int. Society for Music Information Retrieval Conf.*

Bernt Øksendal (2003). *Stochastic differential equations*. Springer.

Michele Pavon (1989). “Stochastic Control and Nonequilibrium Thermodynamical Systems”. In: *Applied Mathematics and Optimization* 19, pp. 187–202.

William Peebles and Saining Xie (2023). “Scalable Diffusion Models with Transformers”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4195–4205.

Yi Ren, Jinzheng He, Xu Tan, Tao Qin, Zhou Zhao, and Tie-Yan Liu (2020). “Popmag: Pop Music Accompaniment Generation”. In: *Proceedings of the 28th ACM international conference on multimedia*, pp. 1198–1206.

Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck (2018). “A Hierarchical Latent Vector Model for Learning Long-term Structure in Music”. In: *International Conference on Machine Learning*, pp. 4364–4373.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer (2022). “High-resolution Image Synthesis with Latent Diffusion Models”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695.

Dimitri von Rütte, Luca Biggio, Yannic Kilcher, and Thomas Hofmann (2022). “FIGARO: Controllable Music Generation using Learned and Expert Features”. In: *The Eleventh International Conference on Learning Representations*.

Jiaming Song, Chenlin Meng, and Stefano Ermon (2021a). “Denoising Diffusion Implicit Models”. In: *International Conference on Learning Representations*.

Jiaming Song, Qinsheng Zhang, Hongxu Yin, Morteza Mardani, Ming-Yu Liu, Jan Kautz, Yongxin Chen, and Arash Vahdat (2023). “Loss-Guided Diffusion Models for Plug-and-Play Controllable Generation”. In: *International Conference on Machine Learning*.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole (2021b). “Score-based Generative Modeling Through Stochastic Differential Equations”. In: *International Conference on Learning Representations*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu (2023). "Roformer: Enhanced Transformer with Rotary Position Embedding". In: *Neurocomputing*, p. 127063.

Evangelos Theodorou, Jonas Buchli, and Stefan Schaal (2010). "A Generalized Path Integral Control Approach to Reinforcement Learning". In: *The Journal of Machine Learning Research* 11, pp. 3137–3181.

Evangelos A Theodorou (2015). "Nonlinear Stochastic Control and Information Theoretic Dualities: Connections, Interdependencies and Thermodynamic Interpretations". In: *Entropy* 17.5, pp. 3352–3375.

Francisco Vargas, Will Grathwohl, and Arnaud Doucet (2023). "Denoising Diffusion Samplers". In: *arXiv preprint arXiv:2302.13834*.

Ziyu Wang, Ke Chen, Junyan Jiang, Yiyi Zhang, Maoran Xu, Shuqi Dai, Xianbin Gu, and Gus Xia (2020). "Pop909: A Pop-song Dataset for Music Arrangement Generation". In: *arXiv preprint arXiv:2008.07142*.

Shih-Lun Wu and Yi-Hsuan Yang (2023). "MuseMorphose: Full-song and Fine-grained Piano Music Style Transfer with One Transformer VAE". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 31, pp. 1953–1967.

Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang (2017). "MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation". In: *International Society for Music Information Retrieval*.

Li-Chia Yang and Alexander Lerch (2020). "On the Evaluation of Generative Models in Music". In: *Neural Computing and Applications* 32.9, pp. 4773–4784.

Zongyu Yin, Federico Reuben, Susan Stepney, and Tom Collins (2023). "Deep Learning's Shallow Gains: a Comparative Evaluation of Algorithms for Automatic Music Generation". In: *Machine Learning* 112.5, pp. 1785–1822.

Chen Zhang, Yi Ren, Kejun Zhang, and Shuicheng Yan (2023a). "SDMuse: Stochastic Differential Music Editing and Generation via Hybrid Representation". In: *IEEE Transactions on Multimedia*.

Qinsheng Zhang and Yongxin Chen (2021). "Path Integral Sampler: a Stochastic Control Approach for Sampling". In: *arXiv preprint arXiv:2111.15141*.

Qinsheng Zhang, Jiaming Song, Xun Huang, Yongxin Chen, and Ming-Yu Liu (2023b). "DiffCollage: Parallel Generation of Large Content with Diffusion Models". In: *IEEE / CVF Computer Vision and Pattern Recognition Conference*.

Chapter 6

CONCLUSIONS AND FUTURE DIRECTIONS

In this thesis, we have delved into the inference dynamics of various deep learning architectures and proposed methods to improve their reliability. In Chapter 2 and 3, we studied the inference dynamics in standard neural networks with discrete layers, and applied self-consistency and Lipschitz analysis to improve robustness against input perturbations. In Chapter 4 and 5, we leveraged control theoretic tools to shape the inference dynamics of continuous-depth neural networks, including neural ODEs and diffusion models. Chapter 4 shows the advantage of forward invariance analysis over Lipschitz bound analysis, noting its independence from integration time. This feature facilitates the certification of robustness in neural ODEs over extended integration time, akin to neural networks with an increased number of discrete layers. In Chapter 5, we introduced stochastic control tools to shape the inference dynamics of diffusion models. This new approach eliminates the need for training surrogate models to steer diffusion models and provides the state-of-the-art performance in guiding diffusion models to follow non-differentiable rules. Looking ahead, we believe that leveraging control theoretic tools to shape inference dynamics in neural networks presents numerous promising research directions and applications, as elaborated below.

Improving Computational Efficiency. Despite of achieving the state-of-the-art performance in guiding diffusion models to follow non-differentiable rules, our Stochastic Control Guidance algorithm (Chapter 5) suffers from higher computational cost compared to alternative methods because it requires multiple samples at each step. Addressing this computational demand is of significant interest. A promising strategy to mitigate these costs involves policy iteration. Essentially, we are solving an optimal control problem through Monte Carlo estimation. It has been shown that the variance of the path weights will decrease as the control gets closer to the optimal control for path integral control (Thijssen and Kappen, 2015). Therefore, initiating the process with a better preliminary control can reduce the number of samples needed to estimate the optimal control. Our preliminary efforts in this direction, utilizing classifier guidance as a preliminary control, have shown that it is possible to reduce the sample count needed for comparable levels of guidance efficacy. Nonetheless, there remains much room for improvement, including

the adoption of online learning techniques to refine the control policies during the sampling process in diffusion models.

Certified Guarantees in Diffusion Models. Current guidance strategies for diffusion models, including those presented in our work, lack a mechanism to guarantee that the generated content adheres strictly to predefined requirements. This limitation becomes particularly critical in safety-sensitive fields like medical imaging, where it may be essential to certify the fidelity of generated content. A straightforward solution might involve iterative sampling at each step, proceeding to the next step only once the output satisfies the criteria. However, this approach is prohibitively costly in computational terms, underscoring the necessity for more sophisticated and adaptive methods.

Theoretical Analysis. The integration of control-inspired algorithms into deep learning systems requires adaptations to accommodate their high dimensionality and complexity. For example, the SCG algorithm, detailed in Chapter 5, uses a one-step estimation of terminal costs instead of unrolling entire trajectories. This simplification introduces a discrepancy between the theoretically optimal control and the control achieved in practice. Quantifying this gap and evaluating the deviation between the algorithm's induced measure and that of the optimal control will enhance our understanding of the method, improving confidence in its application. Furthermore, analyzing the variance in the estimates of optimal control will allow us to gain insights into the optimal sample size for the practical implementation.

Applications. Throughout this thesis, our evaluation of proposed methods have predominantly centered on image classification, nonlinear control, and symbolic music generation. Nonetheless, in principle, these methods can be applied to other applications. For instance, the FI-ODE framework discussed in Chapter 4 could be used in providing certified guarantees for streaming perception (Li et al., 2020). This scenario involves continuous perception tasks rather than a single perception task on a static image, which is suitable to the neural ODE architecture where prediction is made every step. Similarly, the SCG algorithm introduced in Chapter 5 extends beyond the realm of symbolic music generation. Its capability to enforce diffusion models to follow non-differentiable constraints makes it suitable for diverse fields, including protein design, which demands adherence to specific topological constraints, and astronomy imaging, which often relies on black-box physics simulators.

References

Mengtian Li, Yu-Xiong Wang, and Deva Ramanan (2020). “Towards Streaming Perception”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, pp. 473–488.

Sep Thijssen and HJ Kappen (2015). “Path Integral Control and State-dependent Feedback”. In: *Physical Review E* 91.3, p. 032104.

Appendix A

APPENDIX TO CHAPTER 2

A.1 Inference in the Deconvolutional Generative Model

Generative model

We choose the deconvolutional generative model (DGM) (Nguyen et al., 2018) as the generative feedback in CNN-F. The graphical model of the DGM is shown in Figure 2.2 (middle). The DGM has the same architecture as CNN and generates images from high level to low level. Since low level features usually have higher dimension than high level features, the DGM introduces latent variables at each level to account for uncertainty in the generation process.

Let $y \in \mathbb{R}^K$ be label, K is the number of classes. Let $x \in \mathbb{R}^n$ be image and $h \in \mathbb{R}^m$ be encoded features of x after k convolutional layers. In a DGM with L layers in total, $g(\ell) \in \mathbb{R}^{C\times H\times W}$ denotes generated feature map at layer ℓ, and $z(\ell) \in \mathbb{R}^{C\times H\times W}$ denotes latent variables at layer ℓ. We use z_R and z_P to denote latent variables at a layer followed by ReLU and $\mathrm{MaxPool}$, respectively. In addition, we use $(\cdot)^{(i)}$ to denote the ith entry in a tensor. Let $W(\ell)$ and $b(\ell)$ be the weight and bias parameters at layer ℓ in the DGM. We use $(\cdot)(*^\intercal)$ to denote deconvolutional transpose in deconvolutional layers and $(\cdot)^\intercal$ to denote matrix transpose in fully connected layers. In addition, we use $(\cdot)_\uparrow$ and $(\cdot)_\downarrow$ to denote upsampling and downsampling. The generation process in the DGM is as follows:

$$y \sim p(y) \tag{A.1}$$

$$g(L-1) = W(L)^{\intercal} y \tag{A.2}$$

$$z_P(L-1)^{(i)} \sim \text{Ber}\left(\frac{e^{b(L-1)\cdot g(L-1)_{\uparrow}^{(i)}}}{e^{b(L-1)\cdot g(L-1)_{\uparrow}^{(i)}}+1}\right) \tag{A.3}$$

$$g(L-2) = W(L-1)(*^{\intercal})\{g(L-1)_{\uparrow} \odot z_P(L-1)\} \tag{A.4}$$

$$\vdots$$

$$z_R(\ell)^{(i)} \sim \text{Ber}\left(\frac{e^{b(\ell)\cdot g(\ell)^{(i)}}}{e^{b(\ell)\cdot g(\ell)^{(i)}}+1}\right) \tag{A.5}$$

$$g(\ell-1) = W(\ell)(*^{\intercal})\{z_R(\ell) \odot g(\ell)\} \tag{A.6}$$

$$\vdots$$

$$x \sim \mathcal{N}(g(0), \text{diag}(\sigma^2))\,. \tag{A.7}$$

In the above generation process, we generate all the way to the image level. If we choose to stop at layer k to generate image features h, the final generation step is $h \sim \mathcal{N}(g(k), \text{diag}(\sigma^2))$ instead of (A.7). The joint distribution of latent variables from layer 1 to L conditioning on y is:

$$\begin{aligned} p(\{z(\ell)\}_{\ell=1:L}|y) &= p(z(L)|y)\Pi_{\ell=1}^{L-1} p(z(\ell)|\{z(k)\}_{k\geq\ell}, y) \\ &= \text{Softmax}\left(\sum_{\ell=1}^{L}\langle b(\ell), z(\ell)\odot g(\ell)\rangle\right) \end{aligned} \tag{A.8}$$

where $\text{Softmax}(\eta) = \frac{\exp(\eta)}{\sum_{\eta}\exp(\eta)}$ with $\eta = \sum_{\ell=1}^{L}\langle b(\ell), z(\ell)\odot g(\ell)\rangle$.

Proof for Theorem 2.2.1

In this section, we provide proofs for Theorem 2.2.1. In the proof, we use f to denote the feedforward feature map after convolutional layer in the CNN of the same architecture as the DGM, and use $(\cdot)_a$ to denote layers after nonlinear operators. Let v be the logits output from fully-connected layer of the CNN. Without loss of generality, we consider a DGM that has the following architecture. We list the corresponding feedforward feature maps on the left column:

$$
\begin{array}{rll}
 & & g(0) = W(1)(*^{\intercal})g_a(1) \\
\text{Conv} & f(1) = W(1) * x + b(1) & g_a(1) = g(1) \odot z_R(1) \\
\text{ReLU} & f_a(1) = \sigma_{\text{AdaReLU}}(f(1)) & g(1) = W(2)(*^{\intercal})g_a(2) \\
\text{Conv} & f(2) = W(2) * f_a(1) + b(2) & g_a(2) = g(2)_{\uparrow} \odot z_P(2) \\
\text{Pooling} & f_a(2) = \sigma_{\text{AdaPool}}(f(2)) & g(2) = W(3)^{\intercal} v \\
\text{FC} & v = W(3) f_a(2) &
\end{array}
$$

We prove Theorem 2.2.1 which states that CNN with σ_{AdaReLU} and σ_{AdaPool} is the generative classifier derived from the DGM by proving Lemma A.1.1 first.

Definition A.1.1. σ_{AdaReLU} and σ_{AdaPool} are nonlinear operators that adaptively choose how to activate the feedforward feature map based on the sign of the feedback feature map.

$$
\sigma_{\text{AdaReLU}}(f) = \begin{cases} \sigma_{\text{ReLU}}(f), & \text{if } g \geq 0 \\ \sigma_{\text{ReLU}}(-f), & \text{if } g < 0 \end{cases} \tag{A.9}
$$

$$
\sigma_{\text{AdaPool}}(f) = \begin{cases} \sigma_{\text{MaxPool}}(f), & \text{if } g \geq 0 \\ -\sigma_{\text{MaxPool}}(-f), & \text{if } g < 0 \end{cases} \tag{A.10}
$$

Definition A.1.2 (generative classifier)**.** Let v be the logits output of a CNN, and $p(x, y, z)$ be the joint distribution specified by a generative model. A CNN is a generative classifier of a generative model if $\mathrm{Softmax}(v) = p(y|x, z)$.

Lemma A.1.1. *Let y be the label and x be the image. v is the logits output of the CNN that has the same architecture and parameters as the DGM. $g(0)$ is the generated image from the DGM. α is a constant. $\eta(y, z) = \sum_{\ell=1}^{L} \langle b(\ell), z(\ell) \odot g(\ell) \rangle$. Then we have:*

$$
\alpha y^{\intercal} v = g(0)^{\intercal} x + \eta(y, z)\,. \tag{A.11}
$$

Proof.

$$
\begin{aligned}
& g(0)^\intercal x + \eta(y,z) \\
=& \{W(1)(*^\intercal)\{g(1) \odot z_R(1)\}\}^\intercal x + (z_R(1) \odot g(1))^\intercal b(1) + (z_P(2) \odot g(2)_\uparrow)^\intercal b(2) \\
=& (z_R(1) \odot g(1))^\intercal \{W(1)(*^\intercal)x + b(1)\} + (z_P(2) \odot g(2)_\uparrow)^\intercal b(2) \\
=& g(1)^\intercal (z_R(1) \odot f(1)) + (z_P(2) \odot g(2)_\uparrow)^\intercal b(2) \\
=& \{W(2)(*^\intercal)\{g(2)_\uparrow \odot z_P(2)\}\}^\intercal (z_R(1) \odot f(1)) + (z_P(2) \odot g(2)_\uparrow)^\intercal b(2) \\
=& \{g(2)_\uparrow \odot z_P(2)\}^\intercal \{W(2) * (z_R(1) \odot f(1)) + b(2)\} \\
=& (W(3)^\intercal y)^\intercal_\uparrow \{z_P(2) \odot f(2)\} \\
=& \alpha (W(3)^\intercal y)^\intercal (z_P(2) \odot f(2))_\downarrow \\
=& \alpha y^\intercal W(3)(z_P(2) \odot f(2))_\downarrow \\
=& \alpha y^\intercal v \,.
\end{aligned}
$$

□

Remark. *Lemma A.1.1 shows that logits output from the corresponding CNN of the DGM is proportional to the inner product of generated image and input image plus $\eta(y,z)$. Recall from Equation (A.7), since the DGM assumes x to follow a Gaussian distribution centered at $g(0)$, the inner product between $g(0)$ and x is related to $\log p(x|y,z)$. Recall from Equation (A.8) that conditionoal distribution of latent variables in the DGM is parameterized by $\eta(y,z)$. Using these insights, we can use Lemma A.1.1 to show that CNN performs Bayesian inference in the DGM.*
In the proof, the fully-connected layer applies a linear transformation to the input without any bias added. For fully-connected layer with bias term, we modify $\eta(y,z)$ to $\eta'(y,z)$:

$$\eta'(y,z) = \eta(y,z) + y^\intercal b(3)\,.$$

The logits are computed by

$$v = W(3)(f(2) \odot z(2)) + b(3)\,.$$

Following a very similar proof as of Lemma A.1.1, we can show that

$$\alpha y^\intercal v = g^\intercal(0) + \eta'(y,z)\,. \tag{A.12}$$

With Lemma A.1.1, we can prove Theorem 2.2.1. Here, we repeat the theorem and the assumptions on which it is defined:

Assumption 2.2.1. (Constancy assumption in the DGM)

A. The generated image $g(k)$ at layer k of DGM satisfies $||g(k)||_2^2 = \text{const.}$
B. Prior distribution on the label is a uniform distribution: $p(y) = \text{const.}$
C. Normalization factor in $p(z|y)$ for each category is constant: $\sum_z e^{\eta(y,z)} = \text{const.}$

Theorem 2.2.1. Under Assumption 2.2.1, and given a joint distribution $p(h, y, z)$ modeled by the DGM, $p(y|h, z)$ has the same parametric form as a CNN with σ_{AdaReLU} and σ_{AdaPool}.

Proof. Without loss of generality, assume that we generate images at a pixel level. In this case, $h = x$. We use $p(x, y, z)$ to denote the joint distribution specified by the DGM. In addition, we use $q(y|x, z)$ to denote the Softmax output from the CNN, i.e., $q(y|x, z) = \frac{y^\intercal e^v}{\sum_{i=1}^K e^{v^{(i)}}}$. To simplify the notation, we use z instead of $\{z(\ell)\}_{\ell=1:L}$ to denote latent variables across layers.

$$
\begin{aligned}
&\log p(y|x, z) \\
=&\log p(y, x, z) - \log p(x, z) \\
=&\log p(x|y, z) + \log p(z|y) + \log p(y) - \log p(x, z) \\
=&\log p(x|y, z) + \log p(z|y) + \text{const.} && (*) \\
=&-\frac{1}{2\sigma^2}||x - g(0)||_2^2 + \log \text{Softmax}(\eta(y, z)) + \text{const.} \\
=&\frac{1}{\sigma^2} g(0)^\intercal x + \log \text{Softmax}(\eta(y, z)) + \text{const.} && \text{(Assumption 2.2.1.A)} \\
=&\frac{1}{\sigma^2} g(0)^\intercal x + \log \frac{e^{\eta(y,z)}}{\sum_z e^{\eta(y,z)}} + \text{const.} \\
=&\frac{1}{\sigma^2} g(0)^\intercal x + \eta(y, z) + \text{const.} && \text{(Assumption 2.2.1.C)} \\
=&\alpha y^\intercal v + \text{const.} && \text{(Lemma A.1.1)} \\
=&\alpha(\log q(y|x, z) + \log \sum_{i=1}^K e^{v^{(i)}}) + \text{const.} \\
=&\alpha \log q(y|x, z) + \text{const.} && (**)
\end{aligned}
$$

We obtain line $(*)$ for the following reasons: $\log p(y) = \text{const.}$ according to Assumption 2.2.1.B, and $\log p(x, z) = \text{const.}$ because only y is variable, x and z are given. We obtained line $(**)$ because given x and z, the logits output are fixed. Therefore,

$\log \sum_{i=1}^{K} e^{v^{(i)}} = \text{const.}$. Take exponential on both sides of the above equation, we have:

$$p(y|x,z) = \beta q(y|x,z) \tag{A.13}$$

where β is a scale factor. Since both $q(y|x,z)$ and $p(y|x,z)$ are distributions, we have $\sum_y p(y|x,z) = 1$ and $\sum_y q(y|x,z) = 1$. Summing over y on both sides of Equation (A.13), we have $\beta = 1$. Therefore, we have $q(y|x,z) = p(y|x,z)$. □

We have proved that CNN with σ_{AdaReLU} and σ_{AdaPool} is the generative classifier derived from the DGM that generates to layer 0. In fact, we can extend the results to all intermediate layers in the DGM with the following additional assumptions:

Assumption A.1.1. Each generated layer in the DGM has a constant ℓ_2 norm: $||g(\ell)||_2^2 = \text{const.}, \ell = 1, \ldots, L$.

Assumption A.1.2. Normalization factor in $p(z|y)$ up to each layer is constant: $\sum_z e^{\eta(y,\{z(j)\}_{j=\ell:L})} = \text{const.}, \ell = 1, \ldots, L$.

Corollary A.1.1.1. *Under Assumptions A.1.1, A.1.2 and 2.2.1.B, $p(y|f(\ell), \{z(j)\}_{j=\ell:L})$ in the DGM has the same parametric form as a CNN with $\sigma_{AdaReLU}$ and $\sigma_{AdaPool}$ starting at layer ℓ.*

Proof for Proposition 2.2.1.B

In this section, we provide proofs for Proposition 2.2.1.B. In the proof, we inherit the notations that we use for proving Theorem 2.2.1. Without loss of generality, we consider a DGM that has the same architecture as the one we use to prove Theorem 2.2.1.

Proposition 2.2.1.B. Under Assumption 2.2.1, MAP estimate of $z(\ell)$ conditioned on h, y and $\{z(j)\}_{j\neq\ell}$ in the DGM is:

$$\hat{z}_R(\ell) = \mathbb{1}(\sigma_{\text{AdaReLU}}(f(\ell)) \geq 0) \tag{A.14}$$

$$\hat{z}_P(\ell) = \mathbb{1}(g(\ell) \geq 0) \odot \arg\max_{r\times r}(f(\ell)) + \mathbb{1}(g(\ell) < 0) \odot \arg\min_{r\times r}(f(\ell)). \tag{A.15}$$

Proof. Without loss of generality, assume that we generate images at a pixel level. In this case, $h = x$. Then we have

$$
\begin{aligned}
&\arg\max_{z(\ell)} \log p(z(\ell)|\{z(j)\}_{j\neq\ell}, x, y) \\
=&\arg\max_{z(\ell)} \log p(\{z(j)\}_{j=1:L}, x, y) \\
=&\arg\max_{z(\ell)} \log p(x|y, \{z(j)\}_{j=1:L}) + \log p(\{z(j)\}_{j=1:L}|y) + \log p(y) \\
=&\arg\max_{z(\ell)} \log p(x|y, \{z(j)\}_{j=1:L}) + \eta(y, z) + \text{const.} \quad \text{(Assumption 2.2.1.B,C)} \\
=&\arg\max_{z(\ell)} \frac{1}{\sigma^2} g(0)^\intercal x + \eta(y, z) + \text{const.} \quad \text{(Assumption 2.2.1.A)}
\end{aligned}
$$

Using Lemma A.1.1, the MAP estimate of $z_R(\ell)$ is:

$$
\begin{aligned}
\hat{z}_R(\ell) &= \arg\max_{z_R(\ell)} (z_R(\ell) \odot g(\ell))^\intercal f(\ell) \\
&= \mathbb{1}(\sigma_{\text{AdaReLU}}(f(\ell)) \geq 0)\,.
\end{aligned}
$$

The MAP estimate of $z_P(\ell)$ is:

$$
\begin{aligned}
\hat{z}_P(\ell) &= \arg\max_{z_P(\ell)} (z_P(\ell) \odot g(\ell)_\uparrow)^\intercal f(\ell) \\
&= \mathbb{1}(g(\ell) \geq 0) \odot \arg\max_{r\times r}(f(\ell)) + \mathbb{1}(g(\ell) < 0) \odot \arg\min_{r\times r}(f(\ell))\,.
\end{aligned}
$$

□

Incorporating instance normalization in the DGM

Inspired by the constant norm assumptions (Assumptions 2.2.1.A and A.1.1), we incorporate instance normalization into the DGM. We use $\overline{(\cdot)} = \frac{(\cdot)}{||\cdot||_2}$ to denote instance normalization, and $(\cdot)_n$ to denote layers after instance normalization. In this section, we prove that with instance normalization, CNN is still the generative classifier derived from the DGM. Without loss of generality, we consider a DGM that has the following architecture. We list the corresponding feedforward feature maps on the left column:

$$
\begin{array}{rll}
 & & g(0) = W(1)(*^{\intercal}) g_a(1) \\
\text{Conv} & f(1) = W(1) * \overline{x} & g_a(1) = g_n(1) \odot z_R(1) \\
\text{Norm} & f_n(1) = \overline{f(1)} & g_n(1) = \overline{g(1)} \\
\text{ReLU} & f_a(1) = \sigma_{\text{AdaReLU}}(f_n(1) + b(1)) & g(1) = W(2)(*^{\intercal}) g_a(2) \\
\text{Conv} & f(2) = W(2) * f_a(1) & g_a(2) = g_n(2)_{\uparrow} \odot z_P(2) \\
\text{Norm} & f_n(2) = \overline{f(2)} & g_n(2) = \overline{g(2)} \\
\text{Pooling} & f_a(2) = \sigma_{\text{AdaPool}}(f_n(2) + b(2)) & g(2) = W(3)^{\intercal} v \\
\text{FC} & v = W(3) f_a(2) &
\end{array}
$$

Assumption A.1.3. Feedforward feature maps and feedback feature maps have the same ℓ_2 norm:

$$
\begin{aligned}
&||g(\ell)||_2 = ||f(\ell)||_2, \ell = 1, \ldots, L \\
&||g(0)||_2 = ||x||_2 \,.
\end{aligned}
$$

Lemma A.1.2. *Let y be the label and x be the image. v is the logits output of the CNN that has the same architecture and parameters as the DGM. $g(0)$ is the generated image from the DGM, and $\overline{g(0)}$ is normalized $g(0)$ by ℓ_2 norm. α is a constant. $\eta(y, z) = \sum_{\ell=1}^{L} \langle b(\ell), z(\ell) \odot g(\ell) \rangle$. Then we have*

$$
\alpha y^{\intercal} v = \overline{g(0)}^{\intercal} x + \eta(y, z) \,. \tag{A.16}
$$

Proof.

$$
\begin{aligned}
&\overline{g(0)}^{\intercal} x + \eta(y, z) \\
=&\{W(1)(*^{\intercal})\{g_n(1) \odot z_R(1)\}\}^{\intercal} \frac{x}{||g(0)||_2} + (z_R(1) \odot g(1))^{\intercal} b(1) \\
&\quad + (z_P(2) \odot g(2)_{\uparrow})^{\intercal} b(2) \\
=&(z_R(1) \odot g_n(1))^{\intercal} \{W(1)(*^{\intercal})\overline{x}\} + (z_R(1) \odot g(1))^{\intercal} b(1) \\
&\quad + (z_P(2) \odot g(2)_{\uparrow})^{\intercal} b(2) && \text{(Assumption A.1.3)} \\
=&g(1)^{\intercal} \{z_R(1) \odot (f_n(1) + b(1))\} + (z_P(2) \odot g(2)_{\uparrow})^{\intercal} b(2) && \text{(Assumption A.1.3)} \\
=&\{W(2)(*^{\intercal})\{g_n(2)_{\uparrow} \odot z_P(2)\}\}^{\intercal} (z_R(1) \odot f(1)) + (z_P(2) \odot g(2)_{\uparrow})^{\intercal} b(2) \\
=&\{g_n(2)_{\uparrow} \odot z_P(2)\}^{\intercal} \{W(2) * (z_R(1) \odot f(1))\} + (z_P(2) \odot g(2)_{\uparrow})^{\intercal} b(2) \\
=&g(2)^{\intercal} \{z_P(2) \odot (f_n(2) + b(2))\} && \text{(Assumption A.1.3)} \\
=&(W(3)^{\intercal} y)^{\intercal}_{\uparrow} \{z_P(2) \odot f(2)\} \\
=&\alpha (W(3)^{\intercal} y)^{\intercal} (z_P(2) \odot f(2))_{\downarrow} \\
=&\alpha y^{\intercal} W(3) (z_P(2) \odot f(2))_{\downarrow} \\
=&\alpha y^{\intercal} v \,.
\end{aligned}
$$

□

Theorem A.1.3. *Under Assumptions A.1.3 and 2.2.1.B and 2.2.1.C, and given a joint distribution $p(h, y, z)$ modeled by the DGM with instance normalization, $p(y|h, z)$ has the same parametric form as a CNN with $\sigma_{AdaReLU}$, $\sigma_{AdaPool}$ and instance normalization.*

Proof. The proof of Theorem A.1.3 is very similar to that of Theorem 2.2.1 using Lemma A.1.2. Therefore, we omit the detailed proof here. □

Remark. *The instance normalization that we incorporate into the DGM is not the same as the instance normalization that is typically used in image stylization (Ulyanov et al., 2016). The conventional instance normalization computes output y from input x as $y = \frac{x - \mu(x)}{\sigma(x)}$, where μ and σ stands for mean and standard deviation respectively. Our instance normalization does not subtract the mean of the input and divides the input by its ℓ_2 norm to make it have constant ℓ_2 norm.*

CNN-F on ResNet

We can show that CNN-F can be applied to ResNet architecture following similar proofs as above. When there is a skipping connection in the forward pass in ResNet,

we also add a skipping connection in the generative feedback. CNN-F on CNN with and without skipping connections are shown in Figure A.1.

A.2 Additional Experiment Details

Standard Training on Fashion-MNIST

Experimental Setup. We use the following architecture to train CNN-F: Conv2d(32, $3\times$ 3), Instancenorm, AdaReLU, Conv2d(64, $3\times$ 3), Instancenorm, AdaPool, Reshape, FC(128), AdaReLU, FC(10). The instance normalization layer we use is described in Appendix A.1. All the images are scaled between $[-1, +1]$ before training. We train both CNN and CNN-F with Adam (Kingma and Ba, 2015), with weight decay of 0.0005 and learning rate of 0.001.

We train both CNN and CNN-F for 30 epochs using cross-entropy loss and reconstruction loss at pixel level as listed in Table 2.1. The coefficient of cross-entropy loss and reconstruction loss is set to be 1.0 and 0.1 respectively. We use the projected gradient descent (PGD) method to generate adversarial samples within L_∞-norm constraint, and denote the maximum L_∞-norm between adversarial images and clean images as ϵ. The step size in PGD attack is set to be 0.02. Since we preprocess images to be within range $[-1, +1]$, the values of ϵ that we report in this paper are half of their actual values to show a relative perturbation strength with respect to range $[0, 1]$.

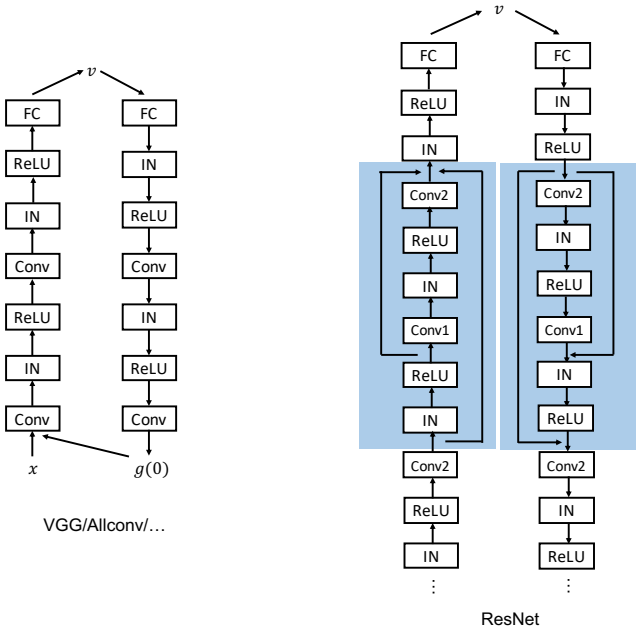
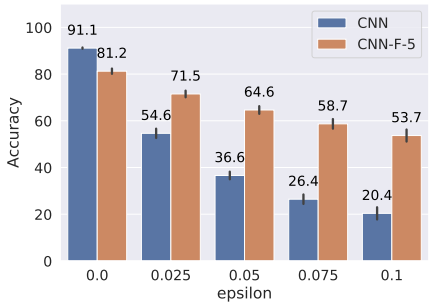
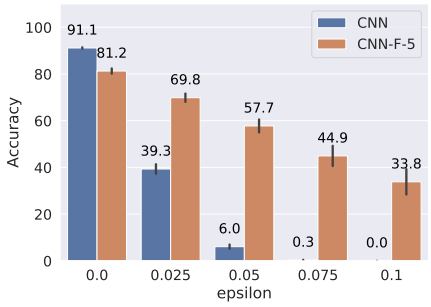


Figure A.1: **CNN-F on CNN with and without skipping connections.**

Adversarial Accuracy against End-to-end Attack. Figure A.2 shows the results of end-to-end (e2e) attack. CNN-F-5 significantly improves the robustness of CNN. Since attacking the first forward pass is more effective than end-to-end attack, we report the adversarial robustness against the former attack in the main text. There are two reasons for the degraded the effectiveness of end-to-end attack. Since σ_{AdaReLU} and σ_{AdaPool} in the CNN-F are non-differentiable, we need to approximate the gradient during back propagation in the end-to-end attack. Furthermore, to perform the end-to-end attack, we need to back propagate through unrolled CNN-F, which is k times deeper than the corresponding CNN, where k is the number of iterations during evaluation.



(a) Standard training. Testing w/ FGSM.



(b) Standard training. Testing w/ PGD-40.

Figure A.2: **Adversarial robustness on Fashion-MNIST against end-to-end attack.** CNN-F-k stands for CNN-F trained with k iterations; PGD-c stands for a PGD attack with c steps. CNN-F achieves higher accuracy on MNIST than CNN for under both standard training and adversarial training. Each accuracy is averaged over 4 runs and the error bar indicates standard deviation.

Adversarial training

Fashion-MNIST. On Fashion-MNIST, we use the following architecture: Conv2d (16, 1×1), Instancenorm, AdaReLU, Conv2d (32, 3×3), Instancenorm, AdaReLU, Conv2d (32, 3×3), Instancenorm, AdaReLU, AdaPool (2×2), Conv2d (64, 3×3), Instancenorm, AdaReLU, AdaPool (2×2), Reshape, FC(1000), AdaReLU, FC(128), AdaReLU, FC(10). The intermediate reconstruction losses are added at the two layers before AdaPool. The coefficients of adversarial sample cross-entropy losses and reconstruction losses are set to 1.0 and 0.1, respectively. We scaled the input images to [-1,+1]. We trained with PGD-7 attack with step size 0.071. We report half of the actual ϵ values in the paper to show a relative perturbation strength with respect to range $[0, 1]$. To train the models, we use SGD optimizer with learning rate of 0.05, weight decay of 0.0005, momentum of 0.9 and gradient clipping with

magnitude of 0.5. The batch size is set to be 256. We use polynomial learning rate scheduling with power of 0.9. We trained the CNN-F models with one iteration for 200 epochs using the following hyper-parameters: online update step size 0.1, ind 2 (using two convolutional layers to encode images to feature space), clean coefficients 0.1.

CIFAR-10. On CIFAR-10, we use Wide ResNet (WRN) (Zagoruyko and Komodakis, 2016) with depth 40 and width 2. The WRN-40-2 architecture consists of 3 network blocks, and each of them consists of 3 basic blocks with 2 convolutional layers. The intermediate reconstruction losses are added at the layers after every network block. The coefficients of adversarial sample cross-entropy losses and reconstruction losses are set to 1.0 and 0.1, respectively. We scaled the input images to [-1,+1]. We trained with PGD-7 attack with step size 0.02. We report half of the actual ϵ values in the paper to show a relative perturbation strength with respect to range $[0, 1]$. To train the models, we use SGD optimizer with learning rate of 0.05, weight decay of 0.0005, momentum of 0.9 and gradient clipping with magnitude of 0.5. The batch size is set to be 256. We use polynomial learning rate scheduling with power of 0.9. We trained the models for 500 epochs with 2 iterations. For the results in Table 2.3, we trained the models using the following hyper-parameters: online update step size 0.1, ind 5, clean coefficients 0.05. In addition, we perform an ablation study on the influence of hyper-parameters.

Which Layer to Reconstruct to? The feature space to reconstruct to in the generative feedback influences the robustness performance of CNN-F. Table A.1 list the adversarial accuracy of CNN-F with different ind configuration, where "ind" stands for the index of the basic block we reconstruct to in the first network block. For instance, ind=3 means that we use all the convolutional layers before and including the third basic block to encode the input image to the feature space. Note that CNN-F models are trained with two iterations, online update step size 0.1, and clean cross-entropy loss coefficient 0.05.

Table A.1: Adversarial accuracy on CIFAR-10 over 3 runs. $\epsilon = 8/255$.

	Clean	PGD (first)	PGD (e2e)	SPSA (first)	SPSA (e2e)	Transfer	Min
CNN-F (ind=3, last)	78.94 ± 0.16	46.03 ± 0.43	60.48 ± 0.66	68.43 ± 0.45	64.14 ± 0.99	65.01 ± 0.65	46.03 ± 0.43
CNN-F (ind=4, last)	78.69 ± 0.57	47.97 ± 0.65	56.40 ± 2.37	69.90 ± 2.04	58.75 ± 3.80	65.53 ± 0.85	47.97 ± 0.65
CNN-F (ind=5, last)	78.68 ± 1.33	$\mathbf{48.90 \pm 1.30}$	49.35 ± 2.55	68.75 ± 1.90	51.46 ± 3.22	66.19 ± 1.37	$\mathbf{48.90 \pm 1.30}$
CNN-F (ind=3, avg)	79.89 ± 0.26	45.61 ± 0.33	$\mathbf{67.44 \pm 0.31}$	68.75 ± 0.66	$\mathbf{70.15 \pm 2.21}$	64.85 ± 0.22	45.61 ± 0.33
CNN-F (ind=4, avg)	80.07 ± 0.52	47.03 ± 0.52	63.59 ± 1.62	70.42 ± 1.42	65.63 ± 1.09	65.92 ± 0.91	47.03 ± 0.52
CNN-F (ind=5, avg)	$\mathbf{80.27 \pm 0.69}$	48.72 ± 0.64	55.02 ± 1.91	$\mathbf{71.56 \pm 2.03}$	58.83 ± 3.72	$\mathbf{67.09 \pm 0.68}$	48.72 ± 0.64

Cross-entropy Loss Coefficient on Clean Images. We find that a larger coefficient

of the cross-entropy loss on clean images tends to produce better end-to-end attack accuracy even though sacrificing the first attack accuracy a bit (Table A.2). When the attacker does not have access to intermediate output of CNN-F, only end-to-end attack is possible. Therefore, one may prefer models with higher accuracy against end-to-end attack. We use cc to denote the coefficients on clean cross-entropy. Note that CNN-F models are trained with one iteration, online update step size 0.1, ind 5.

Table A.2: Adversarial accuracy on CIFAR-10 over 3 runs. $\epsilon = 8/255$.

	Clean	PGD (first)	PGD (e2e)	SPSA (first)	SPSA (e2e)	Transfer	Min
CNN-F (cc=0.5, last)	82.14 ± 0.07	45.98 ± 0.79	59.16 ± 0.99	72.13 ± 2.99	59.53 ± 2.92	67.27 ± 0.35	45.98 ± 0.79
CNN-F (cc=0.1, last)	78.19 ± 0.60	47.65 ± 1.72	56.93 ± 9.20	66.51 ± 1.10	61.25 ± 4.23	64.93 ± 0.70	47.65 ± 1.72
CNN-F (cc=0.05, last)	78.68 ± 1.33	$\mathbf{48.90 \pm 1.30}$	49.35 ± 2.55	68.75 ± 1.90	51.46 ± 3.22	66.19 ± 1.37	$\mathbf{48.90 \pm 1.30}$
CNN-F (cc=0.5, avg)	$\mathbf{83.15 \pm 0.29}$	44.60 ± 0.53	$\mathbf{68.76 \pm 1.04}$	$\mathbf{72.34 \pm 3.54}$	$\mathbf{68.80 \pm 1.18}$	$\mathbf{67.53 \pm 0.48}$	44.60 ± 0.53
CNN-F (cc=0.1, avg)	80.06 ± 0.65	46.77 ± 1.38	63.43 ± 7.77	69.11 ± 0.77	66.25 ± 4.40	65.56 ± 1.08	46.77 ± 1.38
CNN-F (cc=0.05, avg)	80.27 ± 0.69	48.72 ± 0.64	55.02 ± 1.91	71.56 ± 2.03	58.83 ± 3.72	67.09 ± 0.68	48.72 ± 0.64

References

Diederik P. Kingma and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*.

Tan Nguyen, Nhat Ho, Ankit Patel, Anima Anandkumar, Michael I. Jordan, and Richard G. Baraniuk (2018). "A Bayesian Perspective of Convolutional Neural Networks through a Deconvolutional Generative Model". In: *arXiv:1811.02657*.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky (2016). "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *arXiv:1607.08022*.

Sergey Zagoruyko and Nikos Komodakis (2016). "Wide Residual Networks". In: *arXiv:1605.07146*.

Appendix B

APPENDIX TO CHAPTER 3

B.1 Method Details

A Toy Example

In this section, we provide a toy example to walk through the tighter local Lipschitz bound calculation method in a three-layer neural network step by step. Consider a 3-layer neural network with ReLUθ activation:

$$x \to W^1 \to \text{ReLU}\theta \ \to W^2 \to \text{ReLU}\theta \ \to W^3 \to y\,,$$

where input $x \in \mathbb{R}^3$ and output $y \in \mathbb{R}$.

Suppose at a certain training epoch t, weight matrices have the following values,

$$W^1 = W^2 = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}, W^3 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \theta = 1.$$

Consider an input $x = [1, -1, 0]^\top$ and input perturbation $||x' - x|| \leq 0.1$. Directly using the global Lipschitz bound, we have the following result,

$$L_{glob} \leq ||W^3||||W^2||||W^1|| = 9\sqrt{3}. \tag{B.1}$$

Consider our approach for local Lipschitz bound computation. Given input $x = [1, -1, 0]^\top$ and bounded perturbed input $||x' - x|| \leq 0.1$, the feature map after W_1 is $[[2.7, 3.3], [-2.2, -1.8], [-0.1, 0.1]]^\intercal$, where we use $[\text{LB}, \text{UB}]$ to denote the interval bound for every entry. After ReLUθ , the feature map turns into $[1, 0, [0, 0.1]]^\intercal$, where the first entry is a constant because it is clipped by the upper threshold θ, and the second entry is a constant because it is always less than zero. The third entry varies under perturbation. Using these interval bounds, we can compute the indicator matrices I^1_{C}, I^1_θ and I^1_{V}. After the second weight matrix W_2, the feature map is $[3, 0, [0, 0.1]]^\intercal$. We can compute the local indicator matrices at this layer accordingly. Specifically, the local indicator matrices are as follows,

$$I^1_{\text{C}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I^1_\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I^1_{\text{V}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$I_{\mathrm{C}}^2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I_{\theta}^2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, I_{\mathrm{V}}^2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

and the local Lipschitz bound around input $x = [1, -1, 0]^\top$ follows,

$$\begin{aligned} L_{\text{local}}(x) &\leq \|W^3 I_{\mathrm{V}}^2\| \|I_{\mathrm{V}}^2 W^2 I_{\mathrm{V}}^1\| \|I_{\mathrm{V}}^1 W^1\| \\ &= \| \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \| \| \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \| \| \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \| = 1 \end{aligned} \tag{B.2}$$

which is significantly tighter than the global Lipschitz bound obtained in Eq (B.1).

Why Not Consider Linear ReLU Outputs?

Our approach for tighter local Lipschitz bound separate ReLU outputs to two classes: constant ReLU outputs and varying ReLU outputs under perturbation. In fact, there is another case of ReLU outputs, where the outputs of ReLU equal to the inputs. We refer to these outputs as linear ReLU outputs. In this section, we derive the local Lipschitz bound when considering linear ReLU outputs, and we will find this bound is not necessarily smaller than the global Lipshitz bound.

For simplicity, let us consider the standard ReLU activation. There are three different types of ReLU outputs (linear, fixed, varying). We inherit the notations from the main text, and use I_{L} to denote the indicator matrix for linear ReLU outputs. The indicator matrices for the three different types of ReLU outputs are:

$$I_{\mathrm{L}}^l(i,i) = \begin{cases} 1 & \text{if } \mathrm{LB}_i^l > 0 \\ 0 & \text{otherwise} \end{cases}, \quad I_{\mathrm{C}}^l(i,i) = \begin{cases} 1 & \text{if } \mathrm{UB}_i^l \leq 0 \\ 0 & \text{otherwise} \end{cases}, \quad I_{\mathrm{V}}^l = I - I_{\mathrm{L}} - I_{\mathrm{C}}\,, \tag{B.3}$$

where I is the identity matrix.

In addition, we use D_{L} and D_{V} to denote the slope of ReLU outputs for the linear and varying neurons. The elements in D_{L} and D_{V} can either be 0 or 1 due to the piece-wise linear property of ReLU. We omit the constant ReLU outputs here because they are all zero.

$$D_{\mathrm{L}}^l(i,i) = \begin{cases} 1 & \text{if } I_{\mathrm{L}}^l(i,i) = 1 \\ 0 & \text{if } I_{\mathrm{L}}^l(i,i) = 0 \end{cases}, \quad D_{\mathrm{V}}^l(i,i) = \begin{cases} \mathbb{1}(\mathrm{ReLU}(z_i^l) > 0) & \text{if } I_{\mathrm{V}}^l(i,i) = 1 \\ 0 & \text{if } I_{\mathrm{V}}^l(i,i) = 0 \end{cases}, \tag{B.4}$$

where $\mathbb{1}$ denotes an indicator function.

Let us consider a 3-layer neural network with ReLU activation:

$$x \to W^1 \to \text{ReLU} \to W^2 \to \text{ReLU} \to W^3 \to y.$$

Then, the neural network function (denoted as $F(x; W)$) can be written as:

$$\begin{aligned} F(x; W) &= (W^3(D_{\text{L}}^2 + D_{\text{V}}^2)W^2(D_{\text{L}}^1 + D_{\text{V}}^1)W^1)x \\ &= (W^3 D_{\text{L}}^2 W^2 D_{\text{L}}^1 W^1 + W^3 D_{\text{L}}^2 W^2 D_{\text{V}}^1 W^1 + W^3 D_{\text{V}}^2 W^2 D_{\text{L}}^1 W^1 \\ &\quad + W^3 D_{\text{V}}^2 W^2 D_{\text{V}} W^1)x \end{aligned} \tag{B.5}$$

Bounding the local Lipschitz constant from Eq (B.5), we get

$$\begin{aligned} L'_{\text{local}} &= \|W^3 D_{\text{L}}^2 W^2 D_{\text{L}}^1 W^1 + W^3 D_{\text{L}}^2 W^2 D_{\text{V}}^1 W^1 + W^3 D_{\text{V}}^2 W^2 D_{\text{L}}^1 W^1 \\ &\quad + W^3 D_{\text{V}}^2 W^2 D_{\text{V}}^1 W^1\| \\ &\leq \|W^3 D_{\text{L}}^2 W^2 D_{\text{L}}^1 W^1\| + \|W^3 D_{\text{L}}^2 W^2 D_{\text{V}}^1 W^1\| + \|W^3 D_{\text{V}}^2 W^2 D_{\text{L}}^1 W^1\| \\ &\quad + \|W^3 D_{\text{V}}^2 W^2 D_{\text{V}}^1 W^1\| \qquad \text{(B.6)} \\ &\leq \|W^3 D_{\text{L}}^2 W^2 D_{\text{L}}^1 W^1\| + \|W^3 D_{\text{L}}^2 W^2 I_{\text{V}}^1\| \|I_{\text{V}}^1 W^1\| + \|W^3 I_{\text{V}}^2\| \|I_{\text{V}}^2 W^2 D_{\text{L}}^1 W^1\| \\ &\quad + \|W^3 I_{\text{V}}^2\| \|I_{\text{V}}^2 W^2 I_{\text{V}}^1\| \|I_{\text{V}}^1 W^1\|, \qquad \text{(B.7)} \end{aligned}$$

where Eq (B.6) uses triangular inequality, and Eq (B.7) uses Cauchy–Schwarz inequality and the fact that the Lipschitz constant of ReLU is smaller than 1.

The key observation from this approach is that we can "merge" the weight matrices together for linear neurons (the first term in Eq (B.7)). Then we have $\|W^3 D_{\text{L}}^2 W^2 D_{\text{L}}^1 W^1\| \leq \|W^3\| \|W^2\| \|W^1\|$. Furthermore, if the singular vectors for the weight matrices are not aligned, $\|W^3 D_{\text{L}}^2 W^2 D_{\text{L}}^1 W^1\|$ will be much tighter than $\|W^3\| \|W^2\| \|W^1\|$.

However, there are three other non-negative terms in Eq (B.7) from the triangular inequality. Even though the first term could be smaller than the global Lipschitz bound, the summation can be larger than the global Lipschitz bound. In comparison, our approach always guarantees tighter Lipschitz bound as proved in Theorem 3.3.1.

B.2 Review of Other Robust Training Methods

During training, we combine our method with state-of-the-art certifiable training algorithms that involves using L2 Lipchitz bound such as BCP (Lee et al., 2020) and Gloro (Leino et al., 2021). In this section, we first introduce the goal of certifiable robust training and then describe BCP and Gloro in more detail.

Consider a neural network that maps input x to output $z = F(x)$, where $z \in \mathbb{R}^N$. The ground truth label is y. The goal of certifiable training is to minimize the the certified error R. However, computing the exact solution of R is NP-complete (Katz et al., 2017). So in practice, many certifiable training algorithms compute an outer bound of the perturbation sets in logit space $\hat{z}(\mathbb{B}(x))$, and find the worst logit $z^* \in \hat{z}(\mathbb{B}(x))$ to obtain an upper bound of R,

$$\begin{aligned} R &= \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\max_{z\in F(\mathbb{B}_2(x,\epsilon))} \mathbb{1}(\arg\max z \neq y)\big] \\ &\leq \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\max_{z\in \hat{z}(\mathbb{B}(x))} \mathbb{1}(\arg\max z \neq y)\big] \\ &= \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathbb{1}(\arg\max z^* \neq y)] \end{aligned} \tag{B.8}$$

where $\mathbb{B}_2(x,\epsilon)$ denotes the ℓ_2 perturbation set in the input space, $\mathbb{B}_2(x,\epsilon) = \{x' : \|x' - x\|_2 \leq \epsilon\}$, and $F(\mathbb{B}_2(x,\epsilon)) \subset \hat{z}(\mathbb{B}(x))$. In the subsequent text, we use $\mathbb{B}(x)$ in place of $\mathbb{B}_2(x,\epsilon)$ for short. The main difference between BCP and Gloro is how they compute $\hat{z}(\mathbb{B}(x))$ and find the worst logits.

BCP. In BCP, the perturbation set in the input space is propagated through all layers except the last one to get the ball outer bound $\mathbb{B}_2^l$ and box outer bound $\mathbb{B}_\infty^l$ at layer l. Specifically, the l-th layer box outer bound with midpoint m^l and radius r^l is $\mathbb{B}_\infty^l(m^l, r^l) = \{z^l : |z_i^l - m_i^l| \leq r_i^l, \forall i\}$. To compute the indicator matrix in Equation (3.3) and (3.9) for our local Lipschitz bound, we need to bound each neuron by $lb_i^l \leq z_i^l \leq ub_i^l$, where

$$ub^l = \min(m_\infty^l + r_\infty^l, m_{ball}^l + r_{ball}^l) \tag{B.9}$$

$$lb^l = \max(m_\infty^l - r_\infty^l, m_{ball}^l - r_{ball}^l)\,. \tag{B.10}$$

In the case of linear layers, it follows

$$m_\infty^l = W^l m^{l-1} + b^l, r_\infty^l = |W^l| r^{l-1}; m_{ball}^l = W^l z^{l-1} + b^l, {r_{ball}}_i^l = \|W_{i,:}^l\| \rho^{l-1} \tag{B.11}$$

where $\rho^{l-1} = \epsilon \prod_{k=1}^{l-1} \|W^k\|, m^{l-1} = (ub^{l-1} + lb^{l-1})/2$, and $r^{l-1} = (ub^{l-1} - lb^{l-1})/2$.

Then an outer bound of the perturbation sets in logit space $\hat{z}(\mathbb{B}(x))$ is computed via the ball and box constraints on the second last layer: $\hat{z}(\mathbb{B}(x)) = W^L(\mathbb{B}_\infty^{L-1} \cap \mathbb{B}_2^{L-1})$. Finally, the worst logit is computed as

$$z_i^* = F_y(x) - \min_{z\in\hat{z}(\mathbb{B}(x))} (z_y - z_i) \tag{B.12}$$

where $F_y(x)$ denotes the y th entry of $F(x)$ that corresponds to the ground truth label.

Gloro. Unlike BCP, Gloro does not use the local information from box propagation to compute $\hat{z}(\mathbb{B}(x))$ for computation efficiency. In addition, Gloro creates a new class in the logits indicating non-certifiable prediction. The worst logit is computed as by appending the new entry after original logits output $\tilde{z}(x) = [F(x)|\max_{m \neq y} z^*_m]$, where z^*_m is computed by (B.12) with only ball constraints $\hat{z}(\mathbb{B}(x)) = W^L(\mathbb{B}_2^{L-1})$. However, when we combine our method with Gloro, we use their way to construct the new class in the worst logit, but keeps the box constraint when computing $\hat{z}(\mathbb{B}(x))$.

The loss in certifiable training algorithms is a mixed loss function on a normal logit $z = F(x)$ and the worst logit $z^*(x)$:

$$\mathcal{L} = \mathbb{E}_{(x,y)\sim\mathcal{D}}[(1-\lambda)\mathcal{L}(z(x), y) + \lambda\mathcal{L}(z^*(x), y)] \tag{B.13}$$

where $\mathcal{L}$ denotes cross entropy loss, and λ is a hyper-parameter.

B.3 Experimental Details

Training Details

Computing Resources. We train our MNIST and CIFAR models on 1 NVIDIA V100 GPU with 32 GB memory. We train our Tiny-Imagenet model on 4 NVIDIA V100 GPUs.

Architecture. We denote a convolutional layer with output channel c, kernel size k, stride s and output padding p as C(c, k, s, p) and the fully-connected layer with output channel c as F(c). We apply ReLUθ activation after every convolutional layer and fully-connected layer except the last fully-connected layer.

- 4C3F: C(32,3,1,1)-C(32,4,2,1)-C(64,3,1,1)-C(64,4,2,1)-F(512)-F(512)-F(10)
- 6C2F: C(32,3,1,1)-C(32,3,1,1)-C(32,4,2,1)-C(64,3,1,1)-C(64,3,1,1)-C(64,4,2,1)-F(512)-F(10)
- 8C2F: C(64,3,1,1)-C(64,3,1,1)-C(64,4,2,0)-C(128,3,1,1)-C(128,3,1,1)-C(128,4,2,0)-C(256,3,1,1)- C(256,4,2,0)-F(256)-F(200)

Loss. We train with the standard certifiable training loss from Eq (B.13) and the sparsity loss from Eq (3.12) used to encourage constant neurons. The total loss for

ReLU networks is:

$$\begin{aligned}\mathcal{L} = \mathbb{E}_{(x,y)\sim\mathcal{D}}[(1-\lambda)\mathcal{L}(z(x),y) + \lambda\mathcal{L}(z^*(x),y)] \\ + \lambda_{\text{sparsity}}(\max(0, \text{UB}_i^l) \\ + \lambda_\theta \max(0, \theta_i - \text{LB}_i^l))\end{aligned} \tag{B.14}$$

where λ, $\lambda_{\text{sparsity}}$ and λ_θ are hyper-parameters. The total loss for MaxMin networks is defined similarly with $\mathcal{L}_{\text{sparsity}}^{\text{MaxMin}}$ from Eq (3.13).

Hyper-parameters. The hyper-parameters that we use during training is listed in Table B.1. Power iters. stands for the number of power iterations that we use during training. Intial threshold for ReLUθ is the initialization value of the upper threshold in ReLUθ . For all our experiments, we use the Adam optimizer (Kingma and Ba, 2015). For CIFAR experiments, we use the same hyper-parameters for both ReLU and MaxMin activations.

Learning rate scheduling We train with the initial learning rate for m epochs and then start exponential learning rate decay. Let T be the total number of epochs. Learning rate (LR) for epoch t is:

$$\text{LR}(t) = \begin{cases} \text{Initial\_LR} & \text{if } t \leq m \\ \text{Initial\_LR}(\frac{\text{End\_LR}}{\text{Initial\_LR}})^{\frac{t-m}{T-m}} & \text{if } t > m\,. \end{cases} \tag{B.15}$$

ϵ *scheduling* We gradually increase ϵ during training to a target value ϵ_{target} over n epochs. The target value is set to be slightly larger than the ϵ that we aim to certify during evaluation time to give better performance (Leino et al., 2021). ϵ for epoch t is:

$$\epsilon(t) = \begin{cases} \frac{t}{n}\epsilon_{target} & \text{if } t \leq n \\ \epsilon_{target} & \text{if } t > n\,. \end{cases} \tag{B.16}$$

mixture loss scheduling When combined with BCP, we train with the mixed loss of clean cross entropy loss and robust cross entropy loss. We increase λ in Equation B.13 from 0 to 1 linearly over n epochs. λ for epoch t is:

$$\lambda(t) = \begin{cases} \frac{t}{n} & \text{if } t \leq n \\ 1 & \text{if } t > n\,. \end{cases} \tag{B.17}$$

Table B.1: Hyper-parameters used in certifiable training.

Dataset	MNIST	CIFAR	Tiny-Imagenet (ReLU)	Tiny-Imagenet (MaxMin)
Initial LR	0.001	0.001	$2.5\text{e}{-4}$	$1\text{e}{-4}$
End LR	$5\text{e}{-6}$	$1\text{e}{-6}$	$5\text{e}{-7}$	$5\text{e}{-7}$
Batch Size	256	256	128	128
ϵ_{train}	1.58	0.1551	0.16	0.16
λ_{sparse}	0.0	0.0	0.01	0.01
λ_θ	0.0	0.0	0.1	0.1
Warm-up Epochs	0	20	0	0
Total Epochs	300	800	250	250
LR Decay Epoch (m)	150	400	150	150
ϵ Sched. Epochs (n)	150	400	125	125
Power Iters.	5	2	1	3

Ablation Studies

Reproducibility. We train each model with 3 random seeds and report the average accuracy and the standard deviation in Table B.2. For MNIST, we train with our local Lipscthiz bound and the Gloro loss. CIFAR and Tiny-Imagenet, we train with our local Lipscthiz bound and the BCP loss. In the main text (Table 3.2), we report the best accuracy out of 3 runs.

Comparison to Techniques Designed to Certify a Fixed, Post-training Network. We used the state-of-the-art NN verifier (alpha-beta-CROWN (Zhang et al., 2018; Xu et al., 2021; Wang et al., 2021)) from the VNN challenge (Bak et al., 2021) to certify a fixed, post-training network trained. The trained network is trained by techniques proposed by Xiao et al. (2019). Xiao et al. (2019) proposed a ℓ_∞ norm certified defense by imposing ReLU stability regularizer with adversarial training, and verifying the network using a mixed integer linear programming (MILP) verifier. The original approach in Xiao et al. (2019) is not directly applicable to our setting, as they relied on the MILP verifier which cannot scale to the large models evaluated in our paper, and they focused on ℓ_∞ norm robustness. To make a fair comparison to their approach, we made a few extensions to their paper:

Table B.2: Average accuracy and standard deviation over 3 runs. The performance of our method is consistent across different runs.

Data	ϵ	Model	Clean (%)	PGD (%)	Certified (%)
MNIST	1.58	4C3F	96.29 ± 0.07	78.31 ± 0.08	55.79 ± 0.23
CIFAR-10	36/255	4C3F	69.78 ± 0.30	63.95 ± 0.26	53.40 ± 0.08
CIFAR-10	36/255	6C2F	70.64 ± 0.05	64.81 ± 0.05	53.96 ± 0.60
Tiny-Imagenet	36/255	8C2F	29.78 ± 0.08	27.7 ± 0.13	20.5 ± 0.13

- We use ℓ_2 norm adversarial training to replace their ℓ_∞ norm adversarial training.
- We use the same large CIFAR network (4C3F with 62464 neurons) as in our other experiments.
- We use the best NN verifier in the very recent VNN COMP 2021 (Bak et al., 2021), alpha-beta-CROWN (Zhang et al., 2018; Wang et al., 2021), to replace their MILP based verifier.

Additionally, we tried different regularization parameters and reported the best results here. The clean accuray is 57.39%, PGD accuracy is 52.41% and the verified Accuracy is 51.09%. This approach produces a reasonably robust model, thanks to the very recent strong NN verifier. However, its clean, verified and PGD accuracy are worse than ours. Additionally this approach is much less scalable than ours - the verification takes about 150 GPU hours to finish, while our approach takes only 7 minutes to verify the entire dataset.

Comparison to Adversarial Training. We compared our method with adversarial training (AT) (Madry et al., 2018) and TRADES (Zhang et al., 2019) methods as these are known to regularize the Lipschitz constant of neural networks and provide robustness. Although AT and TRADES regularize the neural network Lipschitz compared to a naturally trained neural network, it is still not enough to provide certified robustness. We train with AT and TRADES on CIFAR-10 with the 6C2F architecture and report their accuracies, global and local Lipschitz bounds in Table B.3. The certified accuracy is calculated using local Lipschitz bound. As we can see from the table, although the models trained via AT and TRADES have much smaller Lipschitz bound than naturally trained models (we only use cross entropy loss on clean images in natural training), the Lipschitz bound is still too large to give certified robustness. We also used alpha-beta-CROWN to certify the adversarially trained CIFAR-10 4C3F network. The verified accuracy is still 0%.

Influence of Sparsity Loss on Certified Robustness. To encourage the sparsity of varying ReLU neurons, we design a hinge loss to regularize the neural network (Eq B.14). To study the effectiveness of this sparsity loss, we vary the coefficient of this loss when training a 8C2F model on Tiny-Imagenet. In addition, we find that down-weighting the hinge loss on the upper threshold improves performance. Hence we keep λ_θ as 0.1 for all the models. We report the clean, PGD and certified

Table B.3: Comparison to adversarial training methods on CIFAR-10 with the 6C2F architecture.

Methods	Global Lipschitz bound	Local Lipschitz bound	Clean (%)	PGD (%)	Certified (%)
Standard	1.52×10^9	4.81×10^8	87.7	35.9	0.0
AT	2.27×10^4	1.85×10^4	80.7	70.76	0.0
TRADES	1.82×10^4	1.32×10^4	80.0	72.3	0.0
BCP	11.35	11.08	65.7	60.8	51.3
Ours	7.89	6.68	70.7	64.8	54.3

accuracy in Table B.4. As we can see, too large coefficients on the sparsity loss tends to over-regularize the neural network. When $\lambda_{\text{sparse}} = 0.01$, we obtain the best performance.

Table B.4: Influence of sparsity loss on certified robustness on the TinyImageNet dataset with the 8C2F model.

λ_{sparse}	Clean (%)	PGD (%)	Certified (%)
0.0	30.6	28.3	19.9
0.01	30.8	28.4	20.7
0.03	29.7	27.3	20.2
0.1	28.9	26.7	19.8

Time Cost in Training. Since our method needs to evaluate local Lipschitz bound on every data point during training, we pay additional computational cost than Gloro (Leino et al., 2021) and BCP (Lee et al., 2020). However, our local Lipschitz bound is still much more computational efficient than convex relaxation methods such as CAP (Wong et al., 2018). We report the computation time (s/epoch) in Table B.5.

Table B.5: Comparison of training time per epoch.

		Computation time (s/epoch)			
Data	Model	CAP	Gloro	BCP	Ours
MNIST	4C3F	689	9.0	17.3	45.5
CIFAR-10	4C3F	645	-	23.5	38.2
CIFAR-10	6C2F	1369	6.5	26.0	69.8
Tiny-Imagenet	8C2F	-	-	343.5	398.8

Number of Power Iterations for Convergence. To analyze the computational cost, we plot the histogram of number for power method to converge at each convolutional layer in Figure B.1. We used the 6C2F model on CIFAR-10. Let $u(t)$ be the singular vector computed by power iteration at iteration t, we stop power iteration when $||u(t+1) - u(t)|| \leq 1\text{e}{-3}$.

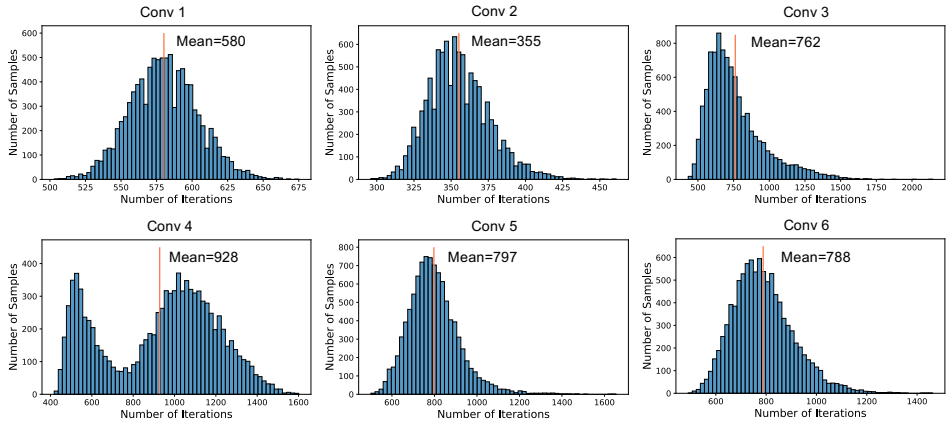


Figure B.1: **Number for power method to converge at each convolutional layer in the 6C2F model.**

Lipschitz bounds and Sparsity of Varying ReLU Outputs during Training. We track the global and Local Lipschitz bound during training for a standard trained CNN, a CNN trained with BCP and global Lipschitz bound, and a CNN trained with BCP and our local Lipschitz bound. All the models are trained with the 6C2F architecture on CIFAR-10. For BCP and our method, we train with $\epsilon_{\text{train}} = 36/255$. We used the hyper-parameters found by BCP (Lee et al., 2020) to train. The total epochs is 200, and first 10 epochs are used for warm-up. We decay learning rate by half every 10 epochs starting from the 121-th epoch. The Lipschitz bound change during training is shown in Figure B.3. Meanwhile, we track the proportion of varying ReLU outputs for all the layers during training in Figure B.2. We can see that the models trained for certified robustness have much fewer varying ReLU outputs than the standard trained model. The sparsity of varying ReLU outputs is desired to tighten our local Lipschitz bound since we can remove more redundant rows and columns in weight matrices.

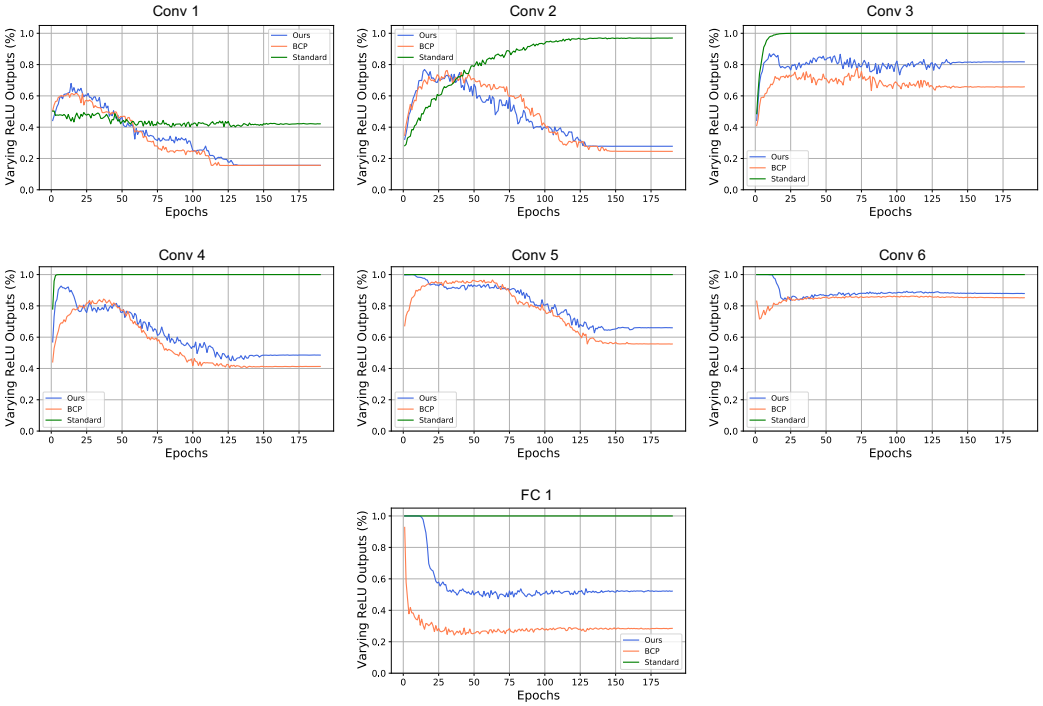


Figure B.2: **Proportion of varying ReLU outputs for all the layers in 6C2F model during training.**

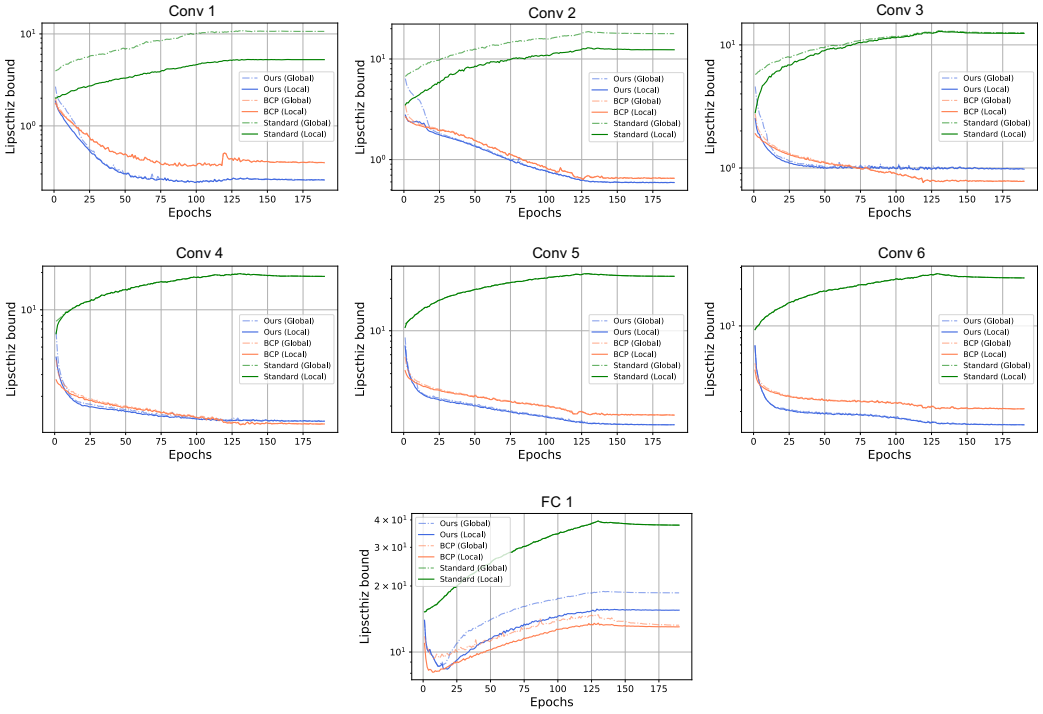


Figure B.3: **Lipschitz bound for all the layers in 6C2F model during training.**

References

Stanley Bak, Changliu Liu, and Taylor Johnson (2021). "The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results". In: *ArXiv preprint*.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer (2017). "Reluplex: An efficient SMT Solver for Verifying Deep Neural Networks". In: *International Conference on Computer Aided Verification (CAV)*.

Diederik P. Kingma and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*.

Sungyoon Lee, Jaewook Lee, and Saerom Park (2020). "Lipschitz-Certifiable Training with a Tight Outer Bound". In: *Advances in Neural Information Processing Systems*.

Klas Leino, Zifan Wang, and Matt Fredrikson (2021). "Globally-Robust Neural Networks". In: *International Conference on Machine Learning*.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu (2018). "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations*.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter (2021). "Beta-CROWN: Efficient Bound Propagation with per-neuron Split Constraints for Neural Network Robustness Verification". In: *Advances in Neural Information Processing Systems*.

Eric Wong, Frank R. Schmidt, Jan Hendrik Metzen, and J. Zico Kolter (2018). "Scaling Provable Adversarial Defenses". In: *Advances in Neural Information Processing Systems*.

Kai Y. Xiao, Vincent Tjeng, Nur Muhammad (Mahi) Shafiullah, and Aleksander Madry (2019). "Training for Faster Adversarial Robustness Verification via Inducing ReLU Stability". In: *International Conference on Learning Representations*.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh (2021). "Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers". In: *International Conference on Learning Representations*.

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan (2019). "Theoretically Principled Trade-off between Robustness and Accuracy". In: *International Conference on Machine Learning*.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel (2018). "Efficient Neural Network Robustness Certification with General Activation Functions". In: *Advances in Neural Information Processing Systems* 31.

Appendix C

APPENDIX TO CHAPTER 4

C.1 Definitions for Class $\mathcal{K}$ Functions

Definition C.1.1 (Class $\mathcal{K}$ Function)**.** A continuous function $\alpha : [0, a) \to [0, \infty)$ for $a \in \mathbb{R}_{>0} \cup \{\infty\}$ belongs to class $\mathcal{K}$ ($a \in \mathcal{K}$) if it satisfies:

1. **Zero at Zero:** $\alpha(0) = 0$
2. **Strictly Increasing:** For all $r_1, r_2 \in [0, a]$ we have that $r_1 < r_2 \Rightarrow \alpha(r_1) < \alpha(r_2)$

Definition C.1.2 (Class $\mathcal{K}_\infty$ Function)**.** A function belongs to $\mathcal{K}_\infty$ if it satisfies:

1. $\alpha \in \mathcal{K}$
2. **Radially Unbounded:** $\lim_{r\to\infty} \alpha(r) = \infty$

Definition C.1.3 (Extended Class $\mathcal{K}^e_\infty$ Function)**.** A continuous function $\alpha : \mathbb{R} \to \mathbb{R}$ belongs to extended $\mathcal{K}^e_\infty$ if it satisfies:

1. **Zero at Zero:** $\alpha(0) = 0$
2. **Strictly Increasing:** For all $r_1, r_2 \in [0, a]$ we have that $r_1 < r_2 \Rightarrow \alpha(r_1) < \alpha(r_2)$

Definition C.1.4 (Class $\mathcal{KL}$ Function)**.** A continuous function $\beta : [0, a) \times [0, \infty) \to [0, \infty)$ belongs to $\mathcal{KL}$ if it satisfies:

1. **Class $\mathcal{K}$ on first argument:** $\forall s \in [0, \infty)\beta(\cdot, s) \in \mathcal{K}$
2. **Asymptotically 0 on second argument:** $\forall r \in [0, a) \lim_{s\to\infty} \beta(r, s) = 0$

C.2 Forward Invariance on a Probability Simplex

For the purposes of certification and training, it is often useful to make the state space $\mathcal{H}$ be a bounded set, as certifying over unbounded sets is typically intractable. For multi-class classification, a natural choice is the probability simplex. Since we initialize $\boldsymbol{\eta}$ within the simplex, it suffices to render the simplex to be forward invariant. We explicitly constrain the states to a probability simplex using a Control-Barrier Function based Quadratic Program (CBF-QP)[1] (Ames et al., 2016), implemented as a differentiable optimization layer (Agrawal et al., 2019).

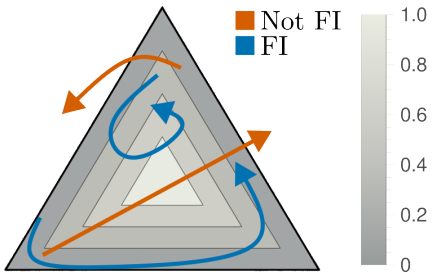


Figure C.1: **The color contours show level-sets of a barrier function in a 3-class probability simplex.**

Barrier functions can be viewed as a variant of Lyapunov functions that only require the state to stay within a set rather than always make progress towards some minimum. Specifically, we choose a potential function h with a 0-super level set (i.e., $\{\boldsymbol{\eta} \in \mathcal{H} | h(\boldsymbol{\eta}) \geq 0\}$) equal to the desired forward invariant set $\mathcal{S}$ (see Figure C.1 for an example). Similarly to the Lyapunov case, there is a point-wise inequality condition that must be true over the forward invariant set:

$$\frac{d}{dt} h(\boldsymbol{\eta}(t)) \geq -\alpha(h(\boldsymbol{\eta})) \tag{C.1}$$

where $\alpha : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a class $\mathcal{K}_\infty$ function. Intuitively, all the flows on the boundary of the forward invariant set must have a positive time-derivative (otherwise there could be a point on the boundary that decreases the value of h and thus exits the forward invariant set). This is the essence of Nagumo's theorem (Nagumo, 1942). Barriers extend this idea with a condition that can be applied everywhere in the target forward invariant set without being overly conservative. As trajectories approach the boundary of the set, Equation (C.1) ensures the time derivative increases until it is positive at the boundary. We use a variation of barrier functions called Control Barrier Functions (CBF). We formalize this concept with Theorem C.3.3.

[1] This is analogous to projected gradient descent (PGD) where we project the dynamics instead of the states.

In our case, the unconstrained dynamics is the output of a neural network and we denote it as $\hat{f}(\boldsymbol{\eta}, \boldsymbol{x})$. To make the dynamics satisfy the barrier conditions, we use a Control Barrier Function Quadratic Program (CBF-QP) Safety Filter (Gurriet et al., 2018):

$$f(\hat{\boldsymbol{f}}) = \underset{\mathtt{f} \in \mathbb{R}^n}{\arg\min} \frac{1}{2} \|\mathtt{f} - \hat{\boldsymbol{f}}\|_2^2 \tag{C.2a}$$

$$\text{s.t} \qquad \mathbb{1}^\top \mathtt{f} = 0 \tag{C.2b}$$

$$\mathtt{f} \geq -\alpha(\boldsymbol{\eta}) \tag{C.2c}$$

where the arguments to the function $\hat{f}$ are omitted for brevity.

Recall that an n-class probability simplex is defined as $\triangle = \{\boldsymbol{\eta} \in \mathbb{R}^n | \sum_{i=1}^n \boldsymbol{\eta}_i = 1, \boldsymbol{\eta}_i \geq 0\}$. Now we show that Equation (C.2b) ensures that the sum of the state stays to be 1 and Equation (C.2c) guarantees the state to be non-negative.

First, we need the sum of $\boldsymbol{\eta}$ stays the same as the initial condition. Taking time derivative of both sides of $\sum_{i=1}^n \boldsymbol{\eta}_i = 1$, we have $\frac{d}{dt}\left(\sum_{i=1}^n \boldsymbol{\eta}_i(t)\right) = \sum_{i=1}^n f_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x})_i = \mathbb{1}^\top f_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x}) = 0$, which is Equation (C.2b). This is natural because the dynamics summing up to zero means the changes from all dimensions summing up to zero, and thus the sum of all dimensions stays the same.

Next, we need each dimension of the state to be non-negative. Since the initial condition has non-negative entries, we just need the set $\{\boldsymbol{\eta} | \boldsymbol{\eta} \geq 0\}$ to be forward invariant. We define forward invariance via barrier functions. For each dimension i, we define $h_i(\boldsymbol{\eta}) = \boldsymbol{\eta}_i$. Then the 0-superlevel set of h_i equals the safe set $\{\boldsymbol{\eta} | \boldsymbol{\eta}_i \geq 0\}$. As long as the condition in Equation (C.1) holds, i.e., $\frac{dh_i}{d\boldsymbol{\eta}} f_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x}) \geq -\alpha(h_i(\boldsymbol{\eta}))$ for some class $\mathcal{K}_\infty$ function α, the set $\{\boldsymbol{\eta} | \boldsymbol{\eta}_i \geq 0\}$ is forward invariant. Plugging in $h_i(\boldsymbol{\eta})$, we have $f_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x}) \geq -\alpha(\boldsymbol{\eta})$, which is Equation (C.2c).

To learn in this setting we differentiate through the QP layer using the KKT conditions as shown in (Agrawal et al., 2019). Given the simple nature of the QP, we implemented a custom solver that uses binary search to efficiently compute solutions, detailed in the supplymentary materials.

To demonstrate the effectiveness of the CBF-QP layer, we visualize the learned trajectories on the CIFAR-3 dataset (a subset of CIFAR-10 with the first 3 classes) in Figure C.2. Each colored line represents a trajectory of an input image from a specific class. As training progresses, the trajectories are trained to evolve to the correct classes. All the trajectories stay in the simplex, implying that the learned dynamics satisfy the constraints in Equation (C.2b) and Equation (C.2c).

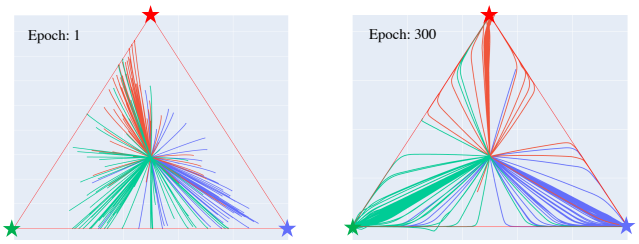


Figure C.2: **Depicting ODE trajectories that satisfy the simplex constraint for CIFAR-3 on epochs 1 and 300.** Each colored line represents the trajectory of an input example of a specific class, and the stars at the corners are colored with the ground-truth class.

C.3 Theorems with Proof

Theorem C.3.1 (ES-CLF Implies Exponential Stability (Ames et al., 2014))**.** *For the ODE in Equations* (4.1a) *and* (4.1b)*, a continuously differentiable function* $V : \mathbb{R}^k \to \mathbb{R}_{\geq 0}$ *is an Exponentially Stabilizing Control Lyapunov Function (ES-CLF) if there are class* $\mathcal{K}_\infty$ *functions* $\overline{\sigma}$ *and* κ *such that:*

$$V(\boldsymbol{\eta}) \leq \overline{\sigma}(\|\boldsymbol{\eta}\|), \tag{C.3}$$

$$\min_{\boldsymbol{\theta}\in\Theta} \left[\frac{\partial V}{\partial \boldsymbol{\eta}}^\top \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x}) + \kappa(V(\boldsymbol{\eta})) \right] \leq 0 \tag{C.4}$$

holds for all $\boldsymbol{\eta} \in \mathcal{R} \subseteq H$ *and* $t \in [0, 1]$*. The existence of an ES-CLF implies that there is a* $\boldsymbol{\theta} \in \Theta$ *that can achieve:*

$$\frac{\partial V}{\partial \boldsymbol{\eta}}^\top \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x}) + \kappa(V(\boldsymbol{\eta})) \leq 0, \tag{C.5}$$

and furthermore the ODE using $\boldsymbol{\theta}$ *is exponentially stable with respect to* V*, i.e.,* $V(\boldsymbol{\eta}(t)) \leq V(\boldsymbol{\eta}(0))e^{-\underline{\kappa}t}$ *for some* $\underline{\kappa} > 0$*.*

Proof. Since Θ is compact, minimums are attained within Θ.
Let $\boldsymbol{\theta}^*(\boldsymbol{\eta}) = \arg\min_{\boldsymbol{\theta}\in\Theta} \frac{\partial V}{\partial \boldsymbol{\eta}}^\top \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x})$. Therefore, from Equation (C.4) we can conclude that:

$$\frac{\partial V}{\partial \boldsymbol{\eta}}^\top \boldsymbol{f}_{\boldsymbol{\theta}^*(\boldsymbol{\eta})}(\boldsymbol{\eta}, \boldsymbol{x}) \leq -\kappa(V(\boldsymbol{\eta})) \qquad \forall \boldsymbol{\eta} \in \mathcal{H} \tag{C.6}$$

For simplicity we will omit the arguments of $\boldsymbol{\theta}^*$. Furthermore, in the case where $\boldsymbol{\theta}^*$ is a set we will select only one. Since $\mathcal{H}$ is compact then

$$\sigma^* = \max_{\boldsymbol{\eta}\in\mathcal{H}} \overline{\sigma}(\|\boldsymbol{\eta}\|). \tag{C.7}$$

This in turn implies that V in bounded in $\mathcal{H}$ which helps us conclude that

$$\overline{V} = \max_{\boldsymbol{\eta} \in \mathcal{H}} V(\boldsymbol{\eta}) \tag{C.8}$$

is well defined which in turn implies

$$\underline{\kappa} = \min_{r \in [0, \overline{V}]} \frac{d\kappa(r)}{dr} \tag{C.9}$$

is well defined. Since κ is strictly increasing, then $\underline{\kappa} > 0$. Notice that $\alpha(r) = \underline{\kappa} r$ satisfies $\alpha \in \mathcal{K}_\infty$ and, by the comparison lemma, $\forall r, \underline{\kappa} r \leq \kappa(r)$. Therefore:

$$\frac{\partial V}{\partial \boldsymbol{\eta}}^\top \boldsymbol{f}_{\boldsymbol{\theta}^*(\boldsymbol{\eta})}(\boldsymbol{\eta}, \boldsymbol{x}) \leq -\kappa(V(\boldsymbol{\eta})) \leq -\underline{\kappa} V(\boldsymbol{\eta}), \qquad \forall \boldsymbol{\eta} \in \mathcal{H} \tag{C.10}$$

In preparation for applying the comparison lemma we will consider the following Initial Value Problem (IVP):

$$y(0) = V(\boldsymbol{\eta}_0) \tag{C.11}$$

$$\dot{y} = -\underline{\kappa} y \tag{C.12}$$

Since this is a linear system solutions for y exist, are unique and take the form $y(t) = V(\boldsymbol{\eta}_0) e^{-\underline{\kappa} t}$. Furthermore, by the comparison lemma we can conclude that:

$$V(\boldsymbol{\eta}(t)) \leq y(t) = V(\boldsymbol{\eta}_0) e^{-\underline{\kappa} t} \tag{C.13}$$

□

Lemma C.3.2 (Solution of Class $\mathcal{K}$ function systems (See Lemma 4.4 in (Khalil, 2002)))**.** *Let $\alpha \in \mathcal{K}_\infty^e$. Then consider the following IVP for $t \in [0, 1]$:*

$$y(0) = y_0 \tag{C.14}$$

$$\dot{y} = -\alpha(y) \tag{C.15}$$

This IVP has unique solutions $y(t) = \beta(y_0, t)$ where $\beta \in \mathcal{KL}$.

Theorem C.3.3 (CBF Existence Implies Forward Invariance (Xu et al., 2015; Nagumo, 1942))**.** *Let the set $\mathcal{S} \subset \mathcal{H}$ be the 0 superlevel set of a continuously differentiable function $h : \mathcal{H} \to \mathbb{R}$, i.e., $\mathcal{S} = \{\boldsymbol{\eta} \in \mathcal{H} | h(\boldsymbol{\eta}) \geq 0\}$. The set $\mathcal{S}$ is forward invariant with respect to the ODE Equations* (4.1a) *and* (4.1b)*, if h is a Control Barrier Function (CBF) i.e., it satisfies either of the following conditions:*

1. *(Xu et al., 2015) There exists a function α for all $\boldsymbol{\eta} \in \mathcal{S}$ so that:*

$$\max_{\boldsymbol{\theta} \in \Theta} \left[\frac{\partial h}{\partial \boldsymbol{\eta}}^\top \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x}) + \alpha(h(\boldsymbol{\eta})) \right] \geq 0, \tag{C.16}$$

   where α is a class $\mathcal{K}_\infty^e$ function (this means $\alpha : \mathbb{R} \to \mathbb{R}$ is strictly increasing and satisfies $\lim_{r \to \infty} \alpha(r) = \infty$).

2. *(Nagumo, 1942) For all $\boldsymbol{\eta} \in \{\boldsymbol{\eta} \in \mathcal{S} | h(\boldsymbol{\eta}) = 0\}$:*

$$\frac{\partial h}{\partial \boldsymbol{\eta}} \neq \mathbf{0} \tag{C.17a}$$

$$\max_{\boldsymbol{\theta} \in \Theta} \left[\frac{\partial h}{\partial \boldsymbol{\eta}}^\top \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x}) \right] \geq 0. \tag{C.17b}$$

Proof. Both conditions follow from fundamentally different arguments. Condition 1 follows from the comparison lemma. Condition 2 uses Nagumo's theorem. In either case we rely on the compactness of Θ to solve the following optimization problem:

$$\boldsymbol{\theta}^*(\boldsymbol{\eta}) = \arg\max_{\boldsymbol{\theta} \in \Theta} \left[\frac{\partial h}{\partial \boldsymbol{\eta}}^\top \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\eta}, \boldsymbol{x}) \right]. \tag{C.18}$$

We will omit the parameters of $\boldsymbol{\theta}^*$ for brevity and choose a random solution in the case where $\boldsymbol{\theta}^*$ returns a set of solutions.

1. Consider the following IVP:

$$y(0) = h(\boldsymbol{\eta}(0)) \tag{C.19}$$

$$\dot{y} = -\alpha(y) \tag{C.20}$$

   which satisfies the conditions of lemma C.3.2. This implies that $y(t)$ is unique and $y(t) = \beta(h(\boldsymbol{\eta}(0)), t)$ where by the assumption of forward invariance $h(\boldsymbol{\eta}(0)) \geq 0$. The by a trivial variant of the comparison lemma we have that $\dot{h}(\boldsymbol{\eta}(t)) \geq -\alpha(h(\boldsymbol{\eta}))$ implies $h(\boldsymbol{\eta}(t)) \geq \beta(h(\boldsymbol{\eta}(0)), t)$ which implies $h(\boldsymbol{\eta}(t)) \geq 0$ for $t \in [0, 1]$.

2. This is a direct application of Nagumo's theorem (Nagumo, 1942).

□

Proof of Theorem 4.3.1

Proof. Define barrier function h as $h = c - V$, then $\mathcal{S}$ is a 0-superlevel set of h. According to Nagumo's theorem (Nagumo, 1942), if $\dot{h}(\boldsymbol{x} + \boldsymbol{\epsilon}; \boldsymbol{\eta}) \geq 0$ on $\partial\mathcal{D} \coloneqq \mathcal{S}$, then $\mathcal{S}$ is forward invariant. Since $\dot{V}(\boldsymbol{\eta}; \boldsymbol{x}) \leq -L_V L_f^x \bar{\epsilon}$ on $\mathcal{D}$, we have $\dot{h}(\boldsymbol{\eta}; \boldsymbol{x}) \geq L_h L_f^x \bar{\epsilon}$ on $\mathcal{D}$ (where L_h is the Lipschitz constant of h and notice that $L_V = L_h$). Then for the perturbed input, we have

$$\dot{h}(\boldsymbol{\eta}; \boldsymbol{x} + \boldsymbol{\epsilon}) = \dot{h}(\boldsymbol{\eta}; \boldsymbol{x}) + \dot{h}(\boldsymbol{\eta}; \boldsymbol{x} + \boldsymbol{\epsilon}) - \dot{h}(\boldsymbol{\eta}; \boldsymbol{x}) \tag{C.21}$$

$$\geq \dot{h}(\boldsymbol{\eta}; \boldsymbol{x}) - \|\dot{h}(\boldsymbol{\eta}; \boldsymbol{x} + \boldsymbol{\epsilon}) - \dot{h}(\boldsymbol{\eta}; \boldsymbol{x})\| \tag{C.22}$$

$$\geq \dot{h}(\boldsymbol{\eta}; \boldsymbol{x}) - L_h L_f^x \bar{\epsilon} \tag{C.23}$$

$$\geq \kappa \underline{V} - L_h L_f^x \bar{\epsilon} \geq 0\,. \tag{C.24}$$

Therefore, $\mathcal{S}$ is still forward invariant for the perturbed inputs with perturbation magnitude smaller than $\bar{\epsilon}$. □

Proof of Theorem 4.3.2

Proof. **(a) Sampling on the level set of a quadratic Lyapunov function.** Consider the Lyapunov function of the form $V(\boldsymbol{\eta}) = \boldsymbol{\eta}^\top P \boldsymbol{\eta}$, where P is a positive definite matrix. Let $\mathcal{D} = \{\boldsymbol{\eta} \in \mathbb{R}^n | \boldsymbol{\eta}^\top P \boldsymbol{\eta} = c\}$ be the c-level set of the Lyapunov function, and let $\mathcal{G}$ be a uniform grid (with spacing r) that covers the Lyapunov level set, i.e., $\min_{\boldsymbol{\eta} \in \mathcal{D}} \boldsymbol{\eta}_i \geq \min_{\boldsymbol{\eta} \in \mathcal{G}} \boldsymbol{\eta}_i$ and $\max_{\boldsymbol{\eta} \in \mathcal{D}} \boldsymbol{\eta}_i \leq \max_{\boldsymbol{\eta} \in \mathcal{G}} \boldsymbol{\eta}_i$, $\forall i = 1, ..., n$. Then $\forall \boldsymbol{\eta} \in \mathcal{D}$, there exists at least a point $\boldsymbol{g} \in \mathcal{G}$ such that $|\boldsymbol{\eta}_i - \boldsymbol{g}_i| \leq \frac{r}{2}$ for all $i = 1, ..., n$. Let $\mathcal{G}_D$ denote those grid points that are close to the decision boundary, i.e., $\mathcal{G}_D = \{\boldsymbol{g} \in \mathcal{G} | |\boldsymbol{\eta}_i - \boldsymbol{g}_i| \leq \frac{r}{2}, \text{ for all } i = 1, ..., n, \forall \boldsymbol{\eta} \in \mathcal{D}\}$.

Now we show that $\mathcal{G}_D \subseteq \{\mathcal{B} \cap \mathcal{G}\}$, where $\mathcal{B}$ is defined in Equation (4.7). Notice that the maximum ℓ_2 distance between a point $\boldsymbol{\eta} \in \mathcal{D}$ and its closest point $\boldsymbol{g} \in \mathcal{G}_D$ is $\frac{\sqrt{n}}{2} r$. Then we find the maximum and minimum Lyapunov function value by perturbing $\boldsymbol{\eta} \in \mathcal{D}$ within $\frac{\sqrt{n}}{2} r$ distance.

Consider the following maximization problem:

$$\max\, (\boldsymbol{\eta} + \boldsymbol{v})^\top P (\boldsymbol{\eta} + \boldsymbol{v}) \tag{C.25a}$$

$$\text{s.t.} \quad \boldsymbol{\eta}^\top P \boldsymbol{\eta} = c \tag{C.25b}$$

$$\boldsymbol{v}^\top \boldsymbol{v} = d^2\,. \tag{C.25c}$$

Let the eigenvalue decomposition of P be $P = U \Lambda U^\top$, where U is an orthonormal matrix, and $\Lambda = \text{diag}\{\lambda_1, \lambda_2, ..., \lambda_n\}$ with $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$. Then the solution

of the above problem equals to the solution of the following problem (rotating the coordinates by U):

$$\max\,(\boldsymbol{\eta}+\boldsymbol{v})^\top\Lambda(\boldsymbol{\eta}+\boldsymbol{v}) \tag{C.26a}$$

$$\text{s.t.}\quad \boldsymbol{\eta}^\top\Lambda\boldsymbol{\eta}=c \tag{C.26b}$$

$$\boldsymbol{v}^\top\boldsymbol{v}=d^2\,. \tag{C.26c}$$

Then we can find the upper bound of the objective by the following:

$$\sum_{i=1}^{n}(\boldsymbol{\eta}_i+\boldsymbol{v}_i)\lambda_i = c+\sum_{i=1}^{n}\lambda_i\boldsymbol{v}_i^2+2\sum_{i=1}^{n}\lambda_i\boldsymbol{\eta}_i\boldsymbol{v}_i \tag{C.27}$$

$$\leq c+\lambda_1 d^2+2\sum_{i=1}^{n}\lambda_i\boldsymbol{\eta}_i\boldsymbol{v}_i \tag{C.28}$$

$$\leq c+\lambda_1 d^2+2\sqrt{\sum_{i=1}^{n}(\sqrt{\lambda_i}\boldsymbol{\eta}_i)^2\sum_{i=1}^{n}(\sqrt{\lambda_i}\boldsymbol{v}_i)^2} \tag{C.29}$$

$$= c+\lambda_1 d^2+2\sqrt{c\sum_{i=1}^{n}\lambda_i\boldsymbol{v}_i^2} \tag{C.30}$$

$$\leq c+\lambda_1 d^2+2d\sqrt{c\lambda_1} \tag{C.31}$$

$$= (\sqrt{c}+d\sqrt{\lambda_1})^2 \coloneqq \overline{c}\,. \tag{C.32}$$

Similarly, we can find the lower bound of the objective by:

$$\sum_{i=1}^{n}(\boldsymbol{\eta}_i+\boldsymbol{v}_i)\lambda_i = c+\sum_{i=1}^{n}\lambda_i\boldsymbol{v}_i^2+2\sum_{i=1}^{n}\lambda_i\boldsymbol{\eta}_i\boldsymbol{v}_i \tag{C.33}$$

$$\geq c+\sum_{i=1}^{n}\lambda_i\boldsymbol{v}_i^2-2\sqrt{c\sum_{i=1}^{n}\lambda_i\boldsymbol{v}_i^2} \tag{C.34}$$

$$= \left(\sqrt{c}-\sqrt{\sum_{i=1}^{n}\lambda_i\boldsymbol{v}_i^2}\right)^2 \tag{C.35}$$

$$\geq (\sqrt{c}-d\sqrt{\lambda_1})^2 \coloneqq \underline{c}\,. \tag{C.36}$$

Therefore, the maximum and minimum Lyapunov function value the points in $\mathcal{G}_D$ can attain are $\overline{c}$ and $\underline{c}$ with $d=\frac{\sqrt{n}}{2}r$, and thus we have $\mathcal{G}_D\subseteq\{\mathcal{B}\cap\mathcal{G}\}$.

By definition of $\mathcal{G}_D$, we have that for any $\boldsymbol{\eta}\in\mathcal{D}$, there exist an $\boldsymbol{s}\in\mathcal{G}_D$ such that $|\boldsymbol{\eta}_i-\boldsymbol{s}_i|\leq\frac{r}{2}$ for all $i=1,...,n$. Since $\mathcal{G}_D\in\{\mathcal{B}\cap\mathcal{G}\}$, we have that for any $\boldsymbol{\eta}\in\mathcal{D}$, there exist an $\boldsymbol{s}\in\{\mathcal{B}\cap\mathcal{G}\}$ such that $|\boldsymbol{\eta}_i-\boldsymbol{s}_i|\leq\frac{r}{2}$ for all $i=1,...,n$.

(b) Sampling on the decision boundary. For any $\boldsymbol{\eta} \in \mathscr{D}_y$, let $\boldsymbol{z} = [N\boldsymbol{\eta}_1, ..., N\boldsymbol{\eta}_n]$. By definition of $\mathscr{D}_y$, in addition to $\sum_i \boldsymbol{z}_i = N$, we have

$$\sum_i \boldsymbol{z}_i = N \tag{C.37}$$

$$\boldsymbol{z}_y = \max_{j \neq y} \boldsymbol{z}_j \,. \tag{C.38}$$

Define $\tilde{\boldsymbol{z}} = [\boldsymbol{z}_1 - \lfloor \boldsymbol{z}_1 \rfloor, ..., \boldsymbol{z}_n - \lfloor \boldsymbol{z}_n \rfloor]$ to be the vector that contains the fractional part of each element in $\boldsymbol{z}$. Then we sort $\tilde{\boldsymbol{z}}$ in a non-decreasing order. For the tied elements that equals to $\tilde{\boldsymbol{z}}_y$, we put $\tilde{\boldsymbol{z}}_y$ as the last. We denote the sorted vector as $\tilde{\boldsymbol{z}}' = [\tilde{\boldsymbol{z}}_{i_1}, ..., \tilde{\boldsymbol{z}}_{i_n}]$, where $\tilde{\boldsymbol{z}}_{i_1} \leq ... \leq \tilde{\boldsymbol{z}}_{i_n}$. Let $v : \mathbb{Z}^+ \to \mathbb{Z}^+$ to be a function that maps the indices in $\tilde{\boldsymbol{z}}$ to the indices in $\tilde{\boldsymbol{z}}'$. For instance, if $\tilde{\boldsymbol{z}}_1$ becomes the third element in $\tilde{\boldsymbol{z}}'$, then $v(1) = 3$. If $\tilde{\boldsymbol{z}}_j = \tilde{\boldsymbol{z}}_y$, we have $v(y) > v(j)$.

Notice that $\sum_i \tilde{\boldsymbol{z}}'_i = \sum_i \tilde{\boldsymbol{z}}_i = \sum_i \boldsymbol{z}_i - \sum_i \lfloor \boldsymbol{z}_i \rfloor = N - \sum_i \lfloor \boldsymbol{z}_i \rfloor$, and $\sum_i \tilde{\boldsymbol{z}}'_i < n$ since $0 \leq \tilde{\boldsymbol{z}}'_i < 1$ for $i = 1, ..., n$. Let $k = n - (N - \sum_i \lfloor \boldsymbol{z}_i \rfloor)$, we have

$$\tilde{\boldsymbol{z}}'_1 + ... + \tilde{\boldsymbol{z}}'_k = (1 - \tilde{\boldsymbol{z}}'_{k+1}) + ... + (1 - \tilde{\boldsymbol{z}}'_n) \,. \tag{C.39}$$

Define vector $\boldsymbol{q}$ as follows:

$$\boldsymbol{q}_{i_j} = \begin{cases} \lfloor N\boldsymbol{\eta}_{i_j} \rfloor, & j = 1, ..., k \\ \lceil N\boldsymbol{\eta}_{i_j} \rceil, & j = k+1, ..., n \,. \end{cases} \tag{C.40}$$

Then we have $|\boldsymbol{q}_i - \boldsymbol{z}_i| < 1$ for all $i = 1, ..., n$. Now we check $\boldsymbol{q}$ satisfies Equation (C.37) and a relaxed version of Equation (C.38). First, we have $\sum_i \boldsymbol{q}_i = N$ because of Equation (C.39). Next, we show $\boldsymbol{q}_y \geq \max_{i \neq y} \boldsymbol{q}_i$ by contradiction. Suppose there exists an index j such that $\boldsymbol{q}_j > \boldsymbol{q}_y$, then it has to be the case where $\lfloor \boldsymbol{z}_j \rfloor = \lfloor \boldsymbol{z}_y \rfloor$ and we take ceiling on $\boldsymbol{z}_j$ and take floor on $\boldsymbol{z}_y$, i.e., $v(j) > k$ and $v(y) \leq k$. This means $\tilde{\boldsymbol{z}}_j > \tilde{\boldsymbol{z}}_y$, because v is the sorted indices of $\tilde{\boldsymbol{z}}$ in a non-decreasing order and this gives $\tilde{\boldsymbol{z}}_j \geq \tilde{\boldsymbol{z}}_y$, and if $\tilde{\boldsymbol{z}}_j = \tilde{\boldsymbol{z}}_y$, we have $v(y) > v(j)$, which is contradictory to $v(j) > k$ and $v(y) \leq k$. Then we have $\boldsymbol{z}_y = \lfloor \boldsymbol{z}_y \rfloor + \tilde{\boldsymbol{z}}_y < \boldsymbol{z}_j = \lfloor \boldsymbol{z}_j \rfloor + \tilde{\boldsymbol{z}}_j$, which is contradictory to Equation (C.38). Therefore, there does not exist a j such that $\boldsymbol{q}_j > \boldsymbol{q}_y$, i.e., $\boldsymbol{q}_y \geq \max_{i \neq y} \boldsymbol{q}_i$. For the cases where $\boldsymbol{q}_y = \max_{i \neq y} \boldsymbol{q}_i$, we have $\boldsymbol{q} \in S_y$, i.e., $\boldsymbol{q}$ is a sampled point.

For the cases where $\boldsymbol{q}_y > \max_{i \neq y} \boldsymbol{q}_i$, we show that we can modify $\boldsymbol{q}$ to $\tilde{\boldsymbol{q}}$ such that $\tilde{\boldsymbol{q}} \in S_y$ and $|\tilde{\boldsymbol{q}}_i - \boldsymbol{z}_i| \leq 1$ for all $i = 1, ..., n$. Let $\mathcal{I} = \{i \in \mathbb{Z}^+ | \boldsymbol{z} \neq y, \boldsymbol{z}_i = \boldsymbol{z}_y\}$ be the set that contains the indices of all runner-up elements in $\boldsymbol{z}$. If $\boldsymbol{q}_y > \max_{i \neq y} \boldsymbol{q}_i$, then we must have $\boldsymbol{q}_y = \lceil N\boldsymbol{\eta}_y \rceil$, and $\boldsymbol{q}_i = \lfloor N\boldsymbol{\eta}_i \rfloor$ for all $i \in \mathcal{I}$. We first let $\tilde{\boldsymbol{q}} = \boldsymbol{q}$,

and then pick an i^* from $\mathcal{I}$. Let $\mathcal{J} = \{j \in \mathbb{Z}^+ | \boldsymbol{q}_j \geq 1, j \neq i^*, j \neq y\}$. Notice that $\boldsymbol{q}_{i^*} + \boldsymbol{q}_y = 2\lfloor \boldsymbol{z}_y \rfloor + 1$, which is an odd number. Since $\sum_i \boldsymbol{q}_i = N$ and N is an even number, $\mathcal{J} \neq \emptyset$. We discuss how to obtain $\tilde{\boldsymbol{q}}$ case by case.

Case 1: If there exists a $j \in \mathcal{J}$ such that $v(j) > k$, we set $\tilde{\boldsymbol{q}}_j = \lfloor N\boldsymbol{\eta}_j \rfloor$, and set $\tilde{\boldsymbol{q}}_{i^*} = \lceil N\boldsymbol{\eta}_{i^*} \rceil$. Then we have $\sum_i \tilde{\boldsymbol{q}} = \sum_{i \neq i^*, i \neq j} \boldsymbol{q}_i + \boldsymbol{q}_{i^*} + 1 + \boldsymbol{q}_j - 1 = N$ and $|\tilde{\boldsymbol{q}}_i - \boldsymbol{z}_i| \leq 1$ for all $i = 1, ..., n$.

Case 2: If $v(j) \leq k$ for all $j \in \mathcal{J}$, there must exist a j such that $\boldsymbol{q}_j < \boldsymbol{q}_{i^*}$. Otherwise, $\boldsymbol{q}_y = \boldsymbol{q}_{i^*} + 1$, and $\boldsymbol{q}_i = \boldsymbol{q}_{i^*}$ for all $i \neq y$. Then $\sum_i \boldsymbol{q}_i = N = n\boldsymbol{q}_{i^*} + 1$, which is contradictory to the assumption that $N \not\equiv 1 (\text{mod } n)$. Then set $\tilde{\boldsymbol{q}}_j = \lceil N\boldsymbol{\eta}_j \rceil$ and $\tilde{\boldsymbol{q}}_y = \lfloor N\boldsymbol{\eta}_y \rfloor$. Since $\boldsymbol{q}_j < \boldsymbol{q}_{i^*}$, we have $\tilde{\boldsymbol{q}}_j = \boldsymbol{q}_j + 1 \leq \tilde{\boldsymbol{q}}_y = \boldsymbol{q}_{i^*}$. □

Remark. *The assumption that $N \not\equiv 1 (\text{mod } n)$ is easy to satisfy. Since we also require N is an even number, as long as n is also an even number, we have $N \not\equiv 1 (\text{mod } n)$. We can also relax this assumption by adding $[\frac{N}{n}, ..., \frac{N}{n}]$ to S_y.*

Custom solver for the CBF-QP

Consider a CBF-QP in the following form:

$$
\begin{aligned}
f(\hat{\boldsymbol{f}}) = \underset{\mathtt{f} \in \mathbb{R}^n}{\arg\min}\ & \frac{1}{2}\|\mathtt{f} - \hat{\boldsymbol{f}}\|_2^2 \\
\text{s.t} \quad & \mathbb{1}^\top \mathtt{f} = b \\
& \underline{\mathtt{f}}(\boldsymbol{\eta}) \leq \mathtt{f} \leq \overline{\mathtt{f}}(\boldsymbol{\eta})
\end{aligned}
\tag{C.41}
$$

where $\underline{\mathtt{f}}$ and $\overline{\mathtt{f}}$ are non-increasing function of $\boldsymbol{\eta}$. By the Karush–Kuhn–Tucker (KKT) conditions, the solution of (C.41) is as follows:

$$
f(\hat{\boldsymbol{f}}) = \left[\hat{\boldsymbol{f}} + \lambda^* \mathbb{1}\right]_{\underline{\mathtt{f}}}^{\overline{\mathtt{f}}} \tag{C.42}
$$

where $[\cdot]_{\underline{\mathtt{f}}}^{\overline{\mathtt{f}}}$ stands for lower and upper clipping by $\underline{\mathtt{f}}$ and $\overline{\mathtt{f}}$, and λ^* is the Lagrangian multiplier. We find λ^* such that $\mathbb{1}^\top f(\hat{\boldsymbol{f}}) = b$ using binary search. Since $f(\hat{\boldsymbol{f}})$ is clipped by $\underline{\mathtt{f}}$ and $\overline{\mathtt{f}}$, the search range of λ^* is $[\min_i(\hat{\boldsymbol{f}}_i - \underline{\mathtt{f}}_i), \max_i(\overline{\mathtt{f}}_i - \hat{\boldsymbol{f}}_i)]$, where $\hat{\boldsymbol{f}}_i$ stands for the ith element in $\hat{\boldsymbol{f}}$, and $\underline{\mathtt{f}}_i$, $\overline{\mathtt{f}}_i$ stand for the ith element in $\underline{\mathtt{f}}(\boldsymbol{\eta})$ and $\overline{\mathtt{f}}(\boldsymbol{\eta})$ respectively. Here we consider a general constraint where there are both lower and upper bounds on $\mathtt{f}$. If there is only a lower bound constraint on $\mathtt{f}$ as in eq. (C.2c), we search λ^* in $[\min_i(\hat{\boldsymbol{f}}_i - \underline{\mathtt{f}}_i), -\min_i \hat{\boldsymbol{f}}_i]$, because if $\lambda^* > -\min_i \hat{\boldsymbol{f}}_i$, then $\mathbb{1}^\top \mathtt{f} > 0$, violating eq. (C.2b).

To differentiate through the solver in training, we derive the derivatives based on the binding conditions of the inequality constraints. First, we define the binding and not binding sets as follows:

$$\mathcal{S} = \{i | f_i = \underline{\mathtt{f}}_i \text{ or } f_i = \overline{\mathtt{f}}_i\}, \quad \mathcal{S}^c = \Omega \backslash S \tag{C.43}$$

$$\mathcal{S}_l = \{i | f_i = \underline{\mathtt{f}}_i\}, \quad \mathcal{S}_l^c = \Omega \backslash \mathcal{S}_l \tag{C.44}$$

$$\mathcal{S}_u = \{i | f_i = \overline{\mathtt{f}}_i\}, \quad \mathcal{S}_u^c = \Omega \backslash \mathcal{S}_u \tag{C.45}$$

where $\Omega = \{i \in \mathbb{Z}^+ | i \leq n\}$. Then the derivatives of f with respect to the inputs $\hat{\boldsymbol{f}}$, $\underline{\mathtt{f}}$ and $\overline{\mathtt{f}}$ are as follows:

$$\frac{df_i}{d\hat{f}_j} = \begin{cases} 0, & i \in \mathcal{S}^c \text{ or } j \in \mathcal{S}^c \\ 1 - \frac{1}{n(\mathcal{S})}, & i = j \in \mathcal{S} \\ -\frac{1}{n(\mathcal{S})} & i \neq j, i \in \mathcal{S}, j \in \mathcal{S} \end{cases} \tag{C.46}$$

$$\frac{df_i}{d\underline{\mathtt{f}}_j} = \begin{cases} 0, & j \in \mathcal{S}_l, \forall i \in \Omega \\ 0, & j \in \mathcal{S}_l^c, i \in \mathcal{S}_l^c \backslash \{j\} \\ 1, & j \in \mathcal{S}_l^c, i = j \\ -\frac{1}{n(\mathcal{S}_l)} & j \in \mathcal{S}_l^c, i \in \mathcal{S}_l \end{cases} \quad \frac{df_i}{d\overline{\mathtt{f}}_j} = \begin{cases} 0, & j \in \mathcal{S}_u, \forall i \in \Omega \\ 0, & j \in \mathcal{S}_u^c, i \in \mathcal{S}_u^c \backslash \{j\} \\ 1, & j \in \mathcal{S}_u^c, i = j \\ -\frac{1}{n(\mathcal{S}_u)} & j \in \mathcal{S}_u^c, i \in \mathcal{S}_u \,. \end{cases} \tag{C.47}$$

Interval Bound Propagation through CBF-QP

The dynamics of our NODE is parameterized by a neural network followed by a CBF-QP layer. Let $\hat{\boldsymbol{f}}(\boldsymbol{\eta})$ be the dynamics output by the neural network, and let $f(\hat{\boldsymbol{f}})$ be the dynamics after CBF-QP layer. Given perturbed input in an interval bound $\underline{\boldsymbol{\eta}_i} \leq \boldsymbol{\eta}_i \leq \overline{\boldsymbol{\eta}_i}$, we first use a a popular linear relaxation based verifier named CROWN (Zhang et al., 2018) to get an interval bound for $\hat{\boldsymbol{f}}$: $\underline{\hat{\boldsymbol{f}}_i} \leq \hat{\boldsymbol{f}}_i \leq \overline{\hat{\boldsymbol{f}}_i}$. However, CROWN does not support perturbation analysis on differentiable optimization layers such as our CBF-QP layer and deriving linear relaxation for CBF-QP can be hard. However, it is possible to derive interval bounds (a special case of linear bounds in CROWN) through CBF-QP. Consider a QP in the form of Equations (C.2a) to (C.2c), we bound each dimension of $f(\hat{\boldsymbol{f}})$ in $\mathcal{O}(n)$ by solving the QP with the corresponding element of the input set to the lower or upper bound (Proposition C.3.1).

Proposition C.3.1. *Consider a CBF-QP in the form of C.41. Define function h_i to be $h_i : \boldsymbol{\eta}, \hat{\boldsymbol{f}} \mapsto f(\hat{\boldsymbol{f}})_i$. Given perturbed input in an interval bound $\underline{\boldsymbol{\eta}_i} \leq \boldsymbol{\eta}_i \leq \overline{\boldsymbol{\eta}_i}$, and $\underline{\hat{\boldsymbol{f}}_i} \leq \hat{\boldsymbol{f}}_i \leq \overline{\hat{\boldsymbol{f}}_i}$, we have*

$$h_i(\boldsymbol{\eta}_{ub}^i, \hat{\boldsymbol{f}}_{lb}^i) \leq f(\hat{\boldsymbol{f}})_i \leq h_i(\boldsymbol{\eta}_{lb}^i, \hat{\boldsymbol{f}}_{ub}^i) \tag{C.48}$$

where $\boldsymbol{\eta}^i_{ub}, \boldsymbol{\eta}^i_{lb}$ *and* $\hat{\boldsymbol{f}}^i_{ub}, \hat{\boldsymbol{f}}^i_{lb}$ *are defined as follows:*

$$\boldsymbol{\eta}^i_{ub} = \begin{cases} \overline{\boldsymbol{\eta}_j}, & j = i \\ \underline{\boldsymbol{\eta}_j}, & j \neq i \end{cases} \qquad \boldsymbol{\eta}^i_{lb} = \begin{cases} \underline{\boldsymbol{\eta}_j}, & j = i \\ \overline{\boldsymbol{\eta}_j}, & j \neq i \end{cases} \tag{C.49}$$

$$\hat{\boldsymbol{f}}^i_{ub} = \begin{cases} \overline{\hat{\boldsymbol{f}}_j}, & j = i \\ \underline{\hat{\boldsymbol{f}}_j}, & j \neq i \end{cases} \qquad \hat{\boldsymbol{f}}^i_{lb} = \begin{cases} \underline{\hat{\boldsymbol{f}}_j}, & j = i \\ \overline{\hat{\boldsymbol{f}}_j}, & j \neq i\,. \end{cases} \tag{C.50}$$

Proof. We prove by contradiction. For the upper bound $f(\hat{\boldsymbol{f}})_i \leq h_i(\boldsymbol{\eta}^i_{lb}, \hat{\boldsymbol{f}}^i_{ub})$, suppose $f(\hat{\boldsymbol{f}})_i > h_i(\boldsymbol{\eta}^i_{lb}, \hat{\boldsymbol{f}}^i_{ub})$. Plug in Equation (C.42), we have

$$\left[\hat{\boldsymbol{f}}_i + \lambda\right]^{\overline{\mathtt{f}}(\boldsymbol{\eta}_i)}_{\underline{\mathtt{f}}(\boldsymbol{\eta}_i)} > \left[\overline{\hat{\boldsymbol{f}}_i} + \lambda'\right]^{\overline{\mathtt{f}}(\underline{\boldsymbol{\eta}_i})}_{\underline{\mathtt{f}}(\overline{\boldsymbol{\eta}_i})}. \tag{C.51}$$

Since $\underline{\mathtt{f}}$ and $\overline{\mathtt{f}}$ are non-increasing function of $\boldsymbol{\eta}$, we have $\underline{\mathtt{f}}(\overline{\boldsymbol{\eta}_i}) \geq \underline{\mathtt{f}}(\boldsymbol{\eta}_i)$ and $\overline{\mathtt{f}}(\underline{\boldsymbol{\eta}_i}) \geq \overline{\mathtt{f}}(\boldsymbol{\eta}_i)$. Then $\hat{\boldsymbol{f}}_i + \lambda > \overline{\hat{\boldsymbol{f}}_i} + \lambda'$. Since $\hat{\boldsymbol{f}}_i \leq \overline{\hat{\boldsymbol{f}}_i}$, we have $\lambda > \lambda'$. Then for all $j \neq i$, we have

$$\left[\hat{\boldsymbol{f}}_j + \lambda\right]^{\overline{\mathtt{f}}(\boldsymbol{\eta}_j)}_{\underline{\mathtt{f}}(\boldsymbol{\eta}_j)} \geq \left[\underline{\hat{\boldsymbol{f}}_j} + \lambda'\right]^{\overline{\mathtt{f}}(\underline{\boldsymbol{\eta}_i})}_{\underline{\mathtt{f}}(\overline{\boldsymbol{\eta}_i})}. \tag{C.52}$$

Sum on both sides of C.51 and C.52, we have

$$\mathbb{1}^\top f(\boldsymbol{\eta}, \hat{\boldsymbol{f}}) > \mathbb{1}^\top f(\boldsymbol{\eta}_{lb}, \hat{\boldsymbol{f}}_{ub}) \tag{C.53}$$

which is contradictory to the equality constraint in C.41. Therefore, we have $f(\hat{\boldsymbol{f}})_i \leq h_i(\boldsymbol{\eta}^i_{lb}, \hat{\boldsymbol{f}}^i_{ub})$. The lower bound in C.48 can be proved in the same way. □

C.4 Sampling Algorithms for Certification

We describe the process of generating samples on the decision boundary in Algorithm 5. The trick is to break down the n-class decision boundary sampling problem to 2 to $(n-1)$-class sampling problems. For instance, to generate samples with k non-zero elements on an n-class decision boundary with density T, one can sample points on an k-class decision boundary with density $T - k$ first, adding 1 to each dimension to make each element non-zero, and assign each element to an n dimensional vector. This operation is denoted by function G in Algorithm 5. G takes two list inputs a and c, increases each element in a by 1, rearranges the elements in a according to the indices given by c, and output a new list w of shape k. Equation C.54 gives the form of the output of G. If the inputs are $y = 0, a = (3, 2, 3, 0), c = \{2, 3, 7\}$ and $k = 8$ (y is the label, a corresponds to a point on 4-class decision boundary, and c

Algorithm 5 Sample the points on the decision boundary by dynamic programming.

Require: Number of classes K, sample density T, solution set sol with dimension $T \times K$.

Initialize each element of sol to be $\emptyset$.
$\mathrm{sol}[0][k] = \{\mathbf{0}_k\}$, where $\mathbf{0}_k = [0, .., 0] \in \mathbb{R}^k$.
$\mathrm{sol}[j][2] = \{[j/2, j/2]\}$.
for j from 2 to T **do**
    for k from 3 to K **do**
        for l from 0 to $k-2$ **do**
            if $j - k + l \geq 0$ and $k - l \geq 0$ **then**
                Let $\mathcal{C}$ be the set that contains $k - l - 1$ combinations of $\{1, 2, ..., k-1\}$.
                $\mathrm{sol}[j][k] = \mathrm{sol}[j][k] \cup \{G(y, a, c, k) | a \in \mathrm{sol}[j-k+l][k-l], c \in \mathcal{C}\}$.
            end if
        end for
    end for
end for

Ensure: $\tilde{S}_y = \mathrm{sol}[T][K]/T$

specifies the non-zero dimension except for the label dimension in an 8-dimensional vector), then the output is $w = (4, 0, 3, 4, 0, 0, 0, 1)$.

$$w_i = \begin{cases} a_0 + 1, & i = y \\ a_{|\{1,2,\cdots,i\} \cap c|} + 1, & i \in c \\ 0. & \text{o/w} \end{cases} \tag{C.54}$$

C.5 Experiment Details

Nonlinear control

Baselines. Although there are many baselines in robust control/RL, the settings and goals are not the same as our work (certified robust forward invariance) and thus not directly comparable (Table C.1). We found the setting and goal in [3] is the most similar to us, and thus we compare with their method in Table 4.1.

Experiment details. The dynamics for the segway system is (Gurriet et al., 2018):

$$\frac{d}{dt}\begin{bmatrix} \phi \\ v \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ \frac{\cos\phi(-1.8u+11.5v+9.8\sin\phi)-10.9u+68.4v-1.2\dot{\phi}^2\sin\phi}{\cos\phi-24.7} \\ \frac{(9.3u-58.8v)\cos\phi+38.6u-234.5v-\sin\phi(208.3+\dot{\phi}^2\cos\phi)}{\cos^2\phi-24.7} \end{bmatrix}. \tag{C.55}$$

All the constants except the acceleration of gravity $g = 9.8$ are system parameters. We enforce robust forward invariance under $\pm 2\%$ perturbation on each of the param-

Paper	Setting	Goal	Certified or not
Robust MPO (Mankowitz et al., 2020)	Continuous MDP. model mis-specification	Maximize worst case RL performance.	No
CROP (Wu et al., 2022)	Discrete MDP. Input state perturbations	Certification of per-state actions and lower bound of cumulative rewards	Yes for action and cumulative reward
Robust MBP (Donti et al., 2020)	Norm-bounded linear differential inclusions. Disturbance is norm-bounded and added to the dynamics	Train a nonlinear controller that satisfies the exponential stability condition under perturbed dynamics	Yes for stability
Robust FI-ODE (ours)	Continuous nonlinear dynamics. Norm-bounded system parameter perturbations.	Train a nonlinear controller that satisfies the forward invariance (safety) condition under perturbed dynamics	Yes for forward invariance (safety)

Table C.1: Settings of baseline methods.

eters. We use a 3 layer MLP as the controller and use the Adam optimizer (Kingma and Ba, 2015). First, we train the controller to imitate a Linear Quadratic Regulator (LQR) controller. Then we jointly learn the Lyapunov function and the controller. We use adversarial training on $\boldsymbol{x}$ and $\boldsymbol{\eta}$ to encourage the smoothness of $\boldsymbol{f_\theta}(\boldsymbol{\eta}, \boldsymbol{x})$ with respect to $\boldsymbol{x}$. Specifically, we find $\boldsymbol{\epsilon}_x$ and $\boldsymbol{\epsilon}_\eta$ that maximizes $\frac{\partial V}{\partial \boldsymbol{\eta}}^\top \boldsymbol{f_\theta}(\boldsymbol{\eta} + \boldsymbol{\epsilon}_\eta, \boldsymbol{x} + \boldsymbol{\epsilon}_x)$ and train on $\boldsymbol{\eta} + \boldsymbol{\epsilon}_\eta$ and $\boldsymbol{x} + \boldsymbol{\epsilon}_x$. We set $\kappa(V(\boldsymbol{\eta}))$ in Equation (4.6) to be a constant: $\kappa(V(\boldsymbol{\eta})) = \kappa' \leq \bar{\epsilon} L_V L_f^x$ (κ' is smaller than the requisite lower bound since we train with adversarial inputs and the requisite lower bound is for nominal inputs). We use learning rate of 0.02 for the Lyapunov function and 0.01 for the controller. Next, we jointly learn the controller and finetune the lyapunov function from the previous stage via adversarial training on both the system states and system parameters. We use learning rate of 0.01 for the controller and 0.002 for the lyapunov function.

To certify forward invariance, we use rejection sampling on the state space to cover the boundary of the forward invariant set. The spacing of the ambient grid is set to 0.01 for all 3 dimensions, and the rejection criteria is Equation (4.7). For *robust* forward invariance, we certify in 2 phases. In phase 1, we set the spacing of the ambient grid to be 0.005, and we also sample a grid on the system parameter space to cover the $\pm 2\%$ perturbation range on the parameters with the spacing in Table C.2. In phase 2, we sample denser states and system parameters around the states that cannot be certified in phase 1. We set the spacing along each state dimension to be 0.0025, and the spacing of the parameters to be those in brackets in Table C.2.

We run all the control experiments on Intel Core i9 CPU. The certification time for forward invariance is 3.1 seconds, while for robust forward invariance, it is 3285.3 seconds. We also report the training time and the standard deviation of forward invariant rate of each method in Table C.3.

Table C.2: Spacing of the sampled grid on the system parameter space in terms of percentage of each parameter value. Spacing for phase 2 is in the bracket.

Parameter	1.8	11.5	10.9	68.4	1.2	9.3	58.8	38.6	234.5	208.3	24.7
Spacing (%)	4 (4)	4 (4)	4 (4)	1 (1)	4 (4)	4 (4)	2 (1)	1 (1)	1 (0.25)	4 (4)	4 (4)

Table C.3: Robustness of controllers trained with different training methods. The numbers are the percentage of trajectories that stay within the forward invariant set under the nominal and adversarial system parameters on 1000 adversarially selected initial states. We report the mean and standard deviation over 3 runs. The certificate column indicates whether or not we can certify the (robust) FI property.

Training Method	**Training Time (s)**	**Empirical FI rate (%)**		**Certificate**	
		Nominal Params	**Adv Params**	**FI**	**Robust FI**
Standard Backprop Training	191.3	58.0 ± 1.9	50.4 ± 1.1	✗	✗
Basic Lyapunov Training (Jimenez Rodriguez et al., 2022)	1.5	90.2 ± 0.3	52.6 ± 0.6	✗	✗
+ Adaptive Sampling	3.3	100 ± 0.0	68.9 ± 0.3	✓	✗
+ Adversarial Training	67.7	100 ± 0.0	97.8 ± 0.3	✓	✗
+ Both (Ours)	96.7	100 ± 0.0	100 ± 0.0	✓	✓

Image classification

Useful techniques for classification problems. Typically the decision boundary $\{\boldsymbol{\eta} \in \mathbb{R}^n | \boldsymbol{\eta}_y = \max_{i \neq y} \boldsymbol{\eta}_i\}$ is not compact on the logit space ($\boldsymbol{\eta} \in \mathbb{R}^n$). However, we need the Lyapunov level set to be compact for certification because we can only sample finite points and verify the conditions hold in their neighborhoods. Therefore, we restrict the states to evolve on a probability simplex for classification problems. To do so, we use a Control-Barrier Function based Quadratic Program (CBF-QP) (Ames et al., 2016), implemented as a differentiable optimization layer (Agrawal et al., 2019) in the dynamics (Appendix C.2). However, the linear relaxation based verifier that we use (CROWN) (Zhang et al., 2018) does not support perturbation analysis on differentiable optimization layers such as our CBF-QP layer. Since deriving linear relaxation for CBF-QP is hard, we derive an interval bound (a special case of linear bounds in CROWN) through CBF-QP (Appendix C.3).

Experiment settings. For image classification tasks, we use orthogonal layers (Trockman and Kolter, 2021) in the neural network so that the dynamics has 1 Lipschitz constant with respect to both the state and the input. Specifically, we have $\hat{\boldsymbol{f}}(\boldsymbol{\eta}, x) = W_3\sigma(W_2\sigma(W_1\boldsymbol{\eta} + g(x)) + b_2) + b_3$, where g is a neural network with 4 orthogonal convolution layers and 3 orthogonal linear layers, and W_1, W_2, W_3 are orthogonal matrices, σ is the ReLU activation function. We set $\kappa(V(\boldsymbol{\eta})) = \bar{\epsilon} L_V L_f^x V(\boldsymbol{\eta})$ in the training loss (Equation (4.6)). Note that on the decision boundary, $V(\boldsymbol{\eta}) = 1$.

In the CBF-QP, we need to pick a class $\mathcal{K}_\infty^e$ function α for the inequality constraint $\mathtt{f} \geq -\alpha(\boldsymbol{\eta})$. Here we use $\alpha(\boldsymbol{\eta}) = c_1(e^{c_2\boldsymbol{\eta}} - 1)$, where $c_1 = 100$, and $c_2 = 0.02$. Comparing with a linear function, this $\alpha(\boldsymbol{\eta})$ leads to a higher margin over Lipschitz ratio, resulting in better certified accuracy.

During training, we train with batch size of 64. For each image, we sample 512 states. From epoch 1 to 10, all the states are uniformly sampled in the simplex. From epoch 11 to epoch 60, we linearly decay the proportion of uniform sampling in the simplex and increase the portion of uniform sampling within the correct classification set for each class. To sample uniformly in the simplex, we first sample n points from exponential distribution $\text{Exp}(1)$ independently, then we normalize the n dimensional vector to have sum 1. To sample uniformly in the correct classification region for each class, we first uniformly sample from the simplex, then we swap the maximum element with the element corresponding to the correct label. We choose κ to be 2.0 in the loss function (eq. (4.6)). We use Adam optimizer with learning rate 0.01, and train for 300 epochs in total.

For certification, we choose $N = 40$ when sampling on the decision boundary. A larger N will lead to better certified accuracy but increases the computational cost dramatically. We ran the experiments on an NVIDIA RTX A6000 GPU.

For baseline methods (Pabbaraju et al., 2020; Chen et al., 2021), the adversarial accuracy is evaluated with PGD attack. For our methods, we use AutoAttack (Croce and Hein, n.d.) to evaluate the empirical adversarial robustness.

Table C.4: Computational costs for certification on CIFAR-10.

Certification Method	**Sampling density (N)**	**# samples**	**Time (s)**	**Certified**
Lipschitz	20	3.67×10^5	1.03	0
Lipschitz	30	5.50×10^6	1.37	27.40
Lipschitz	40	4.13×10^7	2.8	33.46
CROWN	40	4.13×10^7	240	42.27

Computational cost. The main computational costs of our method comes from the number of samples that are needed to cover the boundary of the forward invariant set. Table C.4 compares the computational costs and performance for different certification methods on CIFAR-10. We first compare the results of certifying with Lipschitz bounds and CROWN (Zhang et al., 2018). Certifying with Lipschitz

bounds is faster. Since we can pre-compute the Lipschitz bound of the dynamics, the certification time equals to the inference time on all the states. Certifying with CROWN provides a tighter bound and thus higher certified accuracy but is more computationally expensive than using the Lipschitz bound. We also compare the performance of different sampling density by choosing different N in Equation (4.8). With larger N, we can cover the region of interest with smaller neighborhood around each sampled point. We vary N using the Lipschitz certification method because it is faster to evaluate, but the pattern should remain the same for CROWN. As expected, we get better accuracy with larger sampling density, but the computational time is longer since we have more samples.

References

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter (2019). "Differentiable Convex Optimization Layers". In: *Advances in Neural Information Processing Systems*. Vol. 32.

Aaron D Ames, Kevin Galloway, Koushil Sreenath, and Jessy W Grizzle (2014). "Rapidly Exponentially Stabilizing Control Lyapunov Functions and Hybrid Zero Dynamics". In: *IEEE Transactions on Automatic Control*.

Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada (2016). "Control Barrier Function Based Quadratic Programs for Safety Critical Systems". In: *IEEE Transactions on Automatic Control*.

Tong Chen, Jean B Lasserre, Victor Magron, and Edouard Pauwels (2021). "Semialgebraic Representation of Monotone Deep Equilibrium Models and Applications to Certification". In: *Advances in Neural Information Processing Systems* 34, pp. 27146–27159.

Francesco Croce and Matthias Hein (n.d.). "Reliable Evaluation of Adversarial Robustness with an Ensemble of Diverse Parameter-free Attacks". In: *International Conference on Machine Learning*.

Priya L Donti, Melrose Roderick, Mahyar Fazlyab, and J Zico Kolter (2020). "Enforcing Robust Control Guarantees within Neural Network Policies". In: *International Conference on Learning Representations*.

Thomas Gurriet, Andrew Singletary, Jacob Reher, Laurent Ciarletta, Eric Feron, and Aaron Ames (2018). "Towards a Framework for Realizable Safety Critical Control through Active Set Invariance". In: *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*.

Ivan Dario Jimenez Rodriguez, Aaron D Ames, and Yisong Yue (2022). "LyaNet: A Lyapunov Framework for Training Neural ODEs". In: *International Conference on Machine Learning*.

Hassan K Khalil (2002). “Nonlinear Systems (Third Edition)”. In: *Patience Hall* 115.

Diederik P. Kingma and Jimmy Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations*.

Daniel J Mankowitz, Nir Levine, Rae Jeong, Yuanyuan Shi, Jackie Kay, Abbas Abdolmaleki, Jost Tobias Springenberg, Timothy Mann, Todd Hester, and Martin Riedmiller (2020). “Robust Reinforcement Learning for Continuous Control with Model Misspecification”. In: *International Conference on Learning Representations*.

Mitio Nagumo (1942). “Über Die Lage Der Integralkurven Gewöhnlicher Differentialgleichungen”. In: *Proceedings of the Physico-Mathematical Society of Japan. 3rd Series*.

Chirag Pabbaraju, Ezra Winston, and J Zico Kolter (2020). “Estimating Lipschitz Constants of Monotone Deep Equilibrium Models”. In: *International Conference on Learning Representations*.

Asher Trockman and J Zico Kolter (2021). “Orthogonalizing Convolutional Layers with the Cayley Transform”. In: *International Conference on Learning Representations*.

Fan Wu, Linyi Li, Zijian Huang, Yevgeniy Vorobeychik, Ding Zhao, and Bo Li (2022). “Crop: Certifying Robust Policies For Reinforcement Learning through Functional Smoothing”. In: *International Conference on Learning Representations*.

Xiangru Xu, Paulo Tabuada, Jessy W Grizzle, and Aaron D Ames (2015). “Robustness of Control Barrier Functions for Safety Critical Control”. In: *IFAC-PapersOnLine*.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel (2018). “Efficient Neural Network Robustness Certification with General Activation Functions”. In: *Advances in Neural Information Processing Systems* 31.

Appendix D

APPENDIX TO CHAPTER 5

D.1 Proofs.

Proof of Theorem 5.4.1.

Lemma D.1.1 (Dai Pra, 1991; Pavon, 1989)**.** *The transition probability for the stochastic dynamical system Eq 5.6 with cost Eq 5.7 and optimal control* $\mathbf{u}^*$ *is:*

$$\mathcal{Q}^*_{s,t}(\boldsymbol{\eta}_s, \boldsymbol{\eta}_t) = \mathcal{Q}^0_{s,t}(\boldsymbol{\eta}_s, \boldsymbol{\eta}_t)\frac{\Psi(\boldsymbol{\eta}_t, t)}{\Psi(\boldsymbol{\eta}_s, s)} \tag{D.1}$$

where $\mathcal{Q}^*_{s,t}(\boldsymbol{\eta}_s, \boldsymbol{\eta}_t)$ *denotes the transition probability from state* $\boldsymbol{\eta}_s$ *at time* s *to state* $\boldsymbol{\eta}_t$ *at time* t*, and* $\mathcal{Q}^0_{s,t}(\boldsymbol{\eta}_s, \boldsymbol{\eta}_t)$ *denotes the transition probability of the uncontrolled system Eq 5.5.*

Now we prove Theorem 5.4.1.

Proof. Consider the SDE in Eq 5.6 with initial condition $\boldsymbol{\eta}_0 \sim \delta_{\bar{\boldsymbol{\eta}}_0}$, where $\delta_{\bar{\boldsymbol{\eta}}_0}$ is a Dirac distribution centered at $\bar{\boldsymbol{\eta}}_0$. Define the terminal cost to be $\phi(\boldsymbol{\eta}_T) = \log \frac{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)}{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T|\mathbf{y})}$, where $p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)$ denotes the terminal distribution of the uncontrolled SDE in Eq 5.5 with initial condition $\boldsymbol{\eta}_0 \sim \delta_{\bar{\boldsymbol{\eta}}_0}$, and the target terminal distribution $p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T|\mathbf{y}) := p(\mathbf{y}|\boldsymbol{\eta}_T)p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)/p(\mathbf{y})$. Then we have

$$\Psi(\bar{\boldsymbol{\eta}}_0, 0) = \mathbb{E}_{\mathcal{Q}^0}\left[e^{-\phi(\boldsymbol{\eta}_T)}|\boldsymbol{\eta}_t = \bar{\boldsymbol{\eta}}_0\right] = \int_{\boldsymbol{\eta}_T} \frac{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T|\mathbf{y})}{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)} p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)\mathrm{d}\boldsymbol{\eta}_T = 1 \tag{D.2}$$

$$\Psi(\boldsymbol{\eta}_T, T) = e^{-\phi(\boldsymbol{\eta}_T)} = \frac{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T|\mathbf{y})}{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)}\,. \tag{D.3}$$

Then from Lemma D.1.1, we have

$$\begin{aligned}
\mathcal{Q}^*_{0,T}(\bar{\boldsymbol{\eta}}_0, \boldsymbol{\eta}_T) &= \mathcal{Q}^0_{0,T}(\bar{\boldsymbol{\eta}}_0, \boldsymbol{\eta}_T)\frac{\Psi(\boldsymbol{\eta}_T, T)}{\Psi(\bar{\boldsymbol{\eta}}_0, 0)} \\
&= p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)\frac{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T|\mathbf{y})}{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)} \\
&= p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T|\mathbf{y})\,.
\end{aligned} \tag{D.4}$$

From the properties of reverse-time SDE, we know that if $p_0(\boldsymbol{\eta}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, then $p_T(\boldsymbol{\eta}) = p(\boldsymbol{\eta})$, i.e., $\int p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T) p_0(\bar{\boldsymbol{\eta}}_0) \mathrm{d}\bar{\boldsymbol{\eta}}_0 = p(\boldsymbol{\eta}_T)$. It follows that

$$\begin{aligned}
\mathcal{Q}^*(\boldsymbol{\eta}_T) &= \int \mathcal{Q}^*_{0,T}(\bar{\boldsymbol{\eta}}_0, \boldsymbol{\eta}_T) p_0(\bar{\boldsymbol{\eta}}_0) \mathrm{d}\bar{\boldsymbol{\eta}}_0 \\
&= \int p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T|\mathbf{y}) p_0(\bar{\boldsymbol{\eta}}_0) \mathrm{d}\bar{\boldsymbol{\eta}}_0 \\
&= \int \frac{p(\mathbf{y}|\boldsymbol{\eta}_T) p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)}{p(\mathbf{y})} p_0(\bar{\boldsymbol{\eta}}_0) \mathrm{d}\bar{\boldsymbol{\eta}}_0 \\
&= \frac{p(\mathbf{y}|\boldsymbol{\eta}_T)}{p(\mathbf{y})} \int p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T) p_0(\bar{\boldsymbol{\eta}}_0) \mathrm{d}\bar{\boldsymbol{\eta}}_0 \\
&= \frac{p(\mathbf{y}|\boldsymbol{\eta}_T)}{p(\mathbf{y})} p(\boldsymbol{\eta}_T) \\
&= p(\boldsymbol{\eta}_T|\mathbf{y}) \,. \qquad \text{(D.5)}
\end{aligned}$$

Finally, we show that the optimal control for $\tilde{\phi} = \ell_y(\boldsymbol{\eta}_T)$ is the same as that for $\phi(\boldsymbol{\eta}_T) = \log \frac{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)}{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T|\mathbf{y})}$, because

$$\phi(\boldsymbol{\eta}_T) = \log \frac{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T)}{p_{\bar{\boldsymbol{\eta}}_0}(\boldsymbol{\eta}_T|\mathbf{y})} = \log \frac{p(\mathbf{y})}{p(\mathbf{y}|\boldsymbol{\eta}_T)} = -\log p(\mathbf{y}|\boldsymbol{\eta}_T) + const = \ell_y(\boldsymbol{\eta}_T) + const \,. \qquad \text{(D.6)}$$

The last equality follows from our assumption that $p_0(\mathbf{y}|\boldsymbol{\eta}) \propto e^{-\ell_{\mathbf{y}}(\boldsymbol{\eta})}$. Plugging $\tilde{\phi}(\boldsymbol{\eta}_T)$ and $\phi(\boldsymbol{\eta}_T)$ into Eq 5.13 leads to the same optimal control. □

Proof of Proposition 5.4.1

Proof.

$$\begin{aligned}
\Psi(\boldsymbol{\eta}, t) &= \mathbb{E}_{\mathcal{Q}^0}\left[e^{-\phi(\boldsymbol{\eta}_T)}|\boldsymbol{\eta}_t = \boldsymbol{\eta}\right] \\
&= \mathbb{E}_{\mathcal{Q}^0}\left[p(\mathbf{y}|\boldsymbol{\eta}_T) \cdot c|\boldsymbol{\eta}_t = \boldsymbol{\eta}\right] \\
&= c \cdot \int p(\mathbf{y}|\boldsymbol{\eta}_T) p(\boldsymbol{\eta}_T|\boldsymbol{\eta}_t = \boldsymbol{\eta}) \mathrm{d}\boldsymbol{\eta}_T \qquad \text{(D.7)}
\end{aligned}$$

where $p(\boldsymbol{\eta}_T|\boldsymbol{\eta}_t = \boldsymbol{\eta}) := \mathcal{Q}^0_{t,T}(\boldsymbol{\eta}, \boldsymbol{\eta}_T)$ is the transition probability from state $\boldsymbol{\eta}$ at time t to state $\boldsymbol{\eta}_T$ at time T for the uncontrolled SDE, and it is differentiable with respect to $\boldsymbol{\eta}$ for all $0 \leq t < T$. In addition, notice that in diffusion models,

$$\begin{aligned}
p(\mathbf{y}|\boldsymbol{\eta}_t) &= \int p(\mathbf{y}|\boldsymbol{\eta}_T, \boldsymbol{\eta}_t) p(\boldsymbol{\eta}_T|\boldsymbol{\eta}_t) \mathrm{d}\boldsymbol{\eta}_T \\
&= \int p(\mathbf{y}|\boldsymbol{\eta}_T) p(\boldsymbol{\eta}_T|\boldsymbol{\eta}_t) \mathrm{d}\boldsymbol{\eta}_T \qquad \text{(D.8)}
\end{aligned}$$

where the second equality comes from that $\boldsymbol{\eta}_t$ and $\mathbf{y}$ are conditionally independent given $\boldsymbol{\eta}_T$. Then from Eq D.7, we have $\Psi(\boldsymbol{\eta}_t, t) = c \cdot p(\mathbf{y}|\boldsymbol{\eta}_t)$. □

Remark. *Although $\phi(\boldsymbol{\eta}_T)$ could be non-differentiable when we choose non-differentiable loss functions, the desirability function $\Psi(\boldsymbol{\eta}, t)$ is differentiable with respect to $\boldsymbol{\eta}$ for all $0 \leq t < T$.*

D.2 Compatibility of SCG with Various Sampling Procedures

SCG is compatible with many stochastic sampling procedures in diffusion models. The key of SCG is to sample multiple $\mathbf{x}_{t-1}$ given $\mathbf{x}_t$, and select the one that leads to the sample that follows the rule best. One can choose different sampling procedure to obtain $\mathbf{x}_{t-1}$ given $\mathbf{x}_t$. Algorithm 6 shows how to use SCG with replacement-based editing. Algorithm 7 shows how to use SCG with stochastic DDIM (Song et al., 2021).

Algorithm 6 Editing with SCG.

Require: Encoding of the source music $\tilde{\mathbf{x}}_0$, mask $\mathbf{M}$ (1 for unaltered part and 0 for editing region), noise level K, sampling algorithm (e.g., SCG), desired label $\mathbf{y}$ (optional, do not need if want to create a variant).
$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
$\mathbf{x}_K = \sqrt{\bar{\alpha}_K}\tilde{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_K}\mathbf{z}$
for $t = K$ **to** 1 **do**
  ▷ Estimate the clean sample from noisy sample.
  $\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta\left(\mathbf{x}_t, t\right)\right)$
  ▷ Replacement projection based on the mask.
  $\tilde{\mathbf{x}}_0 = \mathbf{M} \odot \mathbf{x}_0 + (\mathbf{I} - \mathbf{M}) \odot \hat{\mathbf{x}}_0$
  ▷ Predict ϵ from $\tilde{\mathbf{x}}_0$.
  $\tilde{\epsilon} = \frac{1}{\sqrt{1-\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\tilde{\mathbf{x}}_0)$
  ▷ Sampling using corrected ϵ.
  $\mathbf{x}_{t-1} = \mathrm{sampling\_algorithm}(\mathbf{x}_t, t, \epsilon, \mathbf{y})$
end for
return: $\mathbf{x}_0$

D.3 Additional Experiment Results

Unconditional Generation

In Table D.1, we report the overlapping area between the intra-set and inter-set distribution for all 7 musical attributes as proposed in (Yang and Lerch, 2020). The highest and second highest value except for GT are high- lighted in bold and underline respectively. Our method achieves the highest average OA on all the datasets, and achieves the top 2 OA for most of the individual attributes.

Algorithm 7 Stochastic Control Guided stochastic DDIM sampling

Require: Loss function ℓ_y, rule target $\mathbf{y}$, number of samples n, stochasticity $\eta > 0$, number of steps S.
$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $s = S$ **to** 1 **do**
  ▷ Compute the posterior mean of $\mathbf{x}_{\tau_{s-1}}$.
  $\sigma_{\tau_s} = \eta\sqrt{\frac{1-\bar{\alpha}_{\tau_{s-1}}}{1-\bar{\alpha}_{\tau_s}}(1 - \frac{\bar{\alpha}_{\tau_s}}{\bar{\alpha}_{\tau_{s-1}}})}$
  $\hat{\mathbf{x}}_{\tau_{s-1}} = \sqrt{\bar{\alpha}_{\tau_{s-1}}}\left(\frac{\mathbf{x}_{\tau_s} - \sqrt{1-\bar{\alpha}_{\tau_s}}\epsilon_\theta(\mathbf{x}_{\tau_s}, \tau_s)}{\sqrt{\bar{\alpha}_{\tau_s}}}\right) + \sqrt{1 - \bar{\alpha}_{\tau_{s-1}} - \sigma^2_{\tau_s}}\epsilon_\theta\left(\mathbf{x}_{\tau_s}, \tau_s\right)$
  if $s > 1$ **then**
    ▷ Sampling possible next steps.
    $\mathbf{x}^i_{\tau_{s-1}} = \hat{\mathbf{x}}_{\tau_{s-1}} + \sigma_{\tau_s}\mathbf{z}^i$, with $\mathbf{z}^1, ..., \mathbf{z}^n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    ▷ Estimate the clean sample from noisy sample.
    $\hat{\mathbf{x}}^i_0 = \frac{1}{\sqrt{\bar{\alpha}_{\tau_{s-1}}}}\left(\mathbf{x}^i_{\tau_{s-1}} - \sqrt{1 - \bar{\alpha}_{\tau_{s-1}}}\epsilon_\theta\left(\mathbf{x}^i_{\tau_{s-1}}, \tau_{s-1}\right)\right)$
    ▷ Find the direction that minimizes the loss.
    $k = \arg\max_i \log p(\mathbf{y}|\hat{\mathbf{x}}^i_0) = \arg\max_i -\ell_y(\hat{\mathbf{x}}^i_0)$
    $\mathbf{x}_{\tau_{s-1}} = \mathbf{x}^k_{\tau_{s-1}}$
  else
    $\mathbf{x}_{\tau_{s-1}} = \hat{\mathbf{x}}_{\tau_{s-1}}$
  end if
end for
return: $\mathbf{x}_0$

Dataset	Method	Used Pitch	IOI	Pitch Hist	Pitch Range	Velocity	Note Duration	Note Density	Avg
Maestro	GT	0.960 ± 0.007	0.901 ± 0.007	0.980 ± 0.003	0.962 ± 0.004	0.971 ± 0.004	0.884 ± 0.011	0.953 ± 0.005	0.944 ± 0.002
	MusicTr	0.954 ± 0.004	0.896 ± 0.018	0.948 ± 0.011	0.961 ± 0.005	0.967 ± 0.005	0.647 ± 0.022	0.948 ± 0.007	0.903 ± 0.005
	Remi	0.934 ± 0.018	0.794 ± 0.017	0.897 ± 0.010	0.903 ± 0.015	0.906 ± 0.007	0.542 ± 0.021	$\mathbf{0.952 \pm 0.004}$	0.847 ± 0.005
	CPW	0.941 ± 0.012	0.641 ± 0.025	0.866 ± 0.010	0.962 ± 0.005	0.830 ± 0.014	0.436 ± 0.030	0.933 ± 0.007	0.801 ± 0.006
	PolyDiff	0.852 ± 0.006	0.843 ± 0.026	0.888 ± 0.016	0.871 ± 0.006	0.805 ± 0.031	0.777 ± 0.016	0.856 ± 0.005	0.842 ± 0.007
	Ours	$\mathbf{0.961 \pm 0.006}$	$\mathbf{0.901 \pm 0.009}$	$\mathbf{0.960 \pm 0.010}$	$\mathbf{0.963 \pm 0.006}$	$\mathbf{0.971 \pm 0.004}$	$\mathbf{0.910 \pm 0.012}$	0.934 ± 0.005	$\mathbf{0.943 \pm 0.003}$
Muscore	GT	0.959 ± 0.009	0.928 ± 0.019	0.980 ± 0.005	0.965 ± 0.006	0.896 ± 0.008	0.925 ± 0.008	0.963 ± 0.004	0.945 ± 0.004
	MusicTr	0.952 ± 0.012	0.916 ± 0.008	0.891 ± 0.009	0.958 ± 0.005	0.790 ± 0.008	0.851 ± 0.020	$\mathbf{0.949 \pm 0.005}$	0.901 ± 0.004
	Remi	0.924 ± 0.008	$\mathbf{0.917 \pm 0.016}$	0.955 ± 0.010	0.944 ± 0.007	0.731 ± 0.024	0.748 ± 0.028	0.934 ± 0.003	0.879 ± 0.006
	CPW	0.879 ± 0.012	0.839 ± 0.020	0.941 ± 0.008	0.900 ± 0.006	0.750 ± 0.025	0.715 ± 0.030	0.879 ± 0.010	0.843 ± 0.007
	PolyDiff	0.888 ± 0.009	0.831 ± 0.006	0.927 ± 0.012	0.864 ± 0.007	0.693 ± 0.014	0.822 ± 0.015	0.891 ± 0.008	0.845 ± 0.004
	Ours	$\mathbf{0.963 \pm 0.004}$	0.915 ± 0.009	$\mathbf{0.962 \pm 0.009}$	$\mathbf{0.964 \pm 0.004}$	$\mathbf{0.890 \pm 0.006}$	$\mathbf{0.900 \pm 0.012}$	0.946 ± 0.005	$\mathbf{0.934 \pm 0.003}$
Pop	GT	0.956 ± 0.007	0.949 ± 0.006	0.983 ± 0.002	0.955 ± 0.003	0.954 ± 0.004	0.940 ± 0.009	0.963 ± 0.002	0.957 ± 0.002
	MusicTr	0.807 ± 0.016	0.880 ± 0.010	0.852 ± 0.005	0.833 ± 0.011	0.865 ± 0.014	0.871 ± 0.008	0.809 ± 0.011	0.845 ± 0.004
	Remi	0.870 ± 0.014	0.839 ± 0.007	$\mathbf{0.979 \pm 0.002}$	0.827 ± 0.008	0.853 ± 0.013	0.826 ± 0.012	0.867 ± 0.005	0.866 ± 0.004
	CPW	0.921 ± 0.011	0.803 ± 0.022	0.942 ± 0.010	0.927 ± 0.008	0.853 ± 0.006	0.891 ± 0.011	$\mathbf{0.953 \pm 0.008}$	0.899 ± 0.005
	PolyDiff	$\mathbf{0.941 \pm 0.003}$	0.924 ± 0.012	0.964 ± 0.005	$\mathbf{0.937 \pm 0.006}$	0.648 ± 0.007	$\mathbf{0.912 \pm 0.020}$	0.855 ± 0.012	0.883 ± 0.004
	Ours	0.927 ± 0.009	$\mathbf{0.952 \pm 0.004}$	0.969 ± 0.002	0.928 ± 0.013	$\mathbf{0.948 \pm 0.003}$	0.911 ± 0.019	0.941 ± 0.009	$\mathbf{0.939 \pm 0.004}$

Table D.1: Objecetive evaluation of unconditional generation. The overlapping area (OA) for 7 music attributes and the average OA are reported. The highest and second highest OA excluding GT are bolded and underlined respectively.

Editing

Our framework also supports editing. Given an existing music piece, we can modify it within any given time window: either create a new variant or guide it to satisfy new rules. To achieve this, we mainly follow the SDEdit framework (Meng et al., 2022): first we add Gaussian noise of a chosen noise level to the latent music representation

and then progressively remove the noise by reversing the SDE. During the reverse process, we use a mask to distinguish the parts that we want to preserve unaltered and the portion we want to modify, and we condition on the unaltered parts via replacement-based conditioning methods as in (Choi et al., 2021; Kawar et al., 2022). Please refer to Appendix D.2 for the detailed guided editing algorithm.

We benchmark our music editing performance against two estabilished methods: MuseMorphose (Wu and Yang, 2023) and PolyDiffusion (Min et al., 2023). Since these baselines are trained on Pop piano music, we condition on the Pop dataset when evaluating our method. Unlike these baselines, which restrict editing to one specific attribute, our method offers the flexibility to edit any attribute. The editing task involves creating a new music piece that adheres to predefined rules based on an original source music piece (for detailed settings, see Appendix D.4). To assess controllability, we evaluate the error rate between the target and generated attributes. Additionally, we measure the similarity in chroma and groove between the generated piece and the source to gauge their resemblance. The goal is to generate music that not only complies with the desired rules but also closely resembles the original source music.

Table D.2 shows the results. For note density, we experiment with two noise levels: 400 and 500. For chord progression, we use a noise level of 500. We can see that there is a trade-off between controllability and resemblance: higher noise level results in better controllability (lower error) but reduced resemblance (lower similarity metrics).

Rule	Method	Error (%) ↓	Sim_{chr} ↑	Sim_{grv} ↑
Note Density	MuseMorphose	29.34	**0.9130**	**0.9184**
	Ours-400	35.62	0.9119	0.8511
	Ours-500	**27.87**	0.8173	0.7153
Chord Progression	PolyDiffusion	70.48	0.5902	**0.7515**
	Ours	**12.62**	**0.8236**	0.6974

Table D.2: Editing performance. For note density, we experimented with noise level of 400 and 500. For chord progression, we used noise level of 500.

D.4 Detailed Experiment Setup

Music Rules

We consider three music rules and give their definitions below.

Pitch Histogram: We compute the histogram of 12 pitch classes over the whole piano roll. Pitch velocity is considered when computing the histogram. The target $\mathbf{y}$

is a 12-dimensional vector specifying the desired pitch histogram.

Note Density: We control both vertical and horizontal note density. We compute note density within 128×128 windows. For a piano roll of shape 128×1024, the target $\mathbf{y}$ is a 16-dimensional vector, the first 8 dimension are for vertical note density and the last 8 dimension are for horizontal note density.

Vertical note density $\text{ND}_{\text{vertical}}$ is computed by

$$\text{ND}_{\text{vertical}} = \frac{1}{T} \sum_{t=1}^{T} n_{\text{on}}(t) \tag{D.9}$$

where $n_{\text{on}}(t)$ stands for the number of on-notes at time t, and T is the window size, we set $T = 128$.

Horizontal note density $\text{ND}_{\text{horizontal}}$ is computed by

$$\text{ND}_{\text{horizontal}} = \sum_{t=1}^{T} \mathbb{1}(n_{\text{start}}(t) \geq 1) \tag{D.10}$$

where $n_{\text{start}}(t)$ stands for the number of notes that start at time t.

Chord Progression: We extract chords using chord analysis tool from the music21 (Cuthbert and Ariza, 2010) package, and group them into 7 classes. We extract 8 chords in total for the 128×1024 piano roll, each chord is the longest chord within a 128×128 window. The target $\mathbf{y}$ is an 8-dimensional vector specifying the desired chord for each 128×128 window.

Training Setup

Data Augmentation. We use the same data augmentation for both VAE and diffusion model training.

- **Key shift**: Entire piano rolls were shifted by up to 6 pitches, effectively functioning as a key switch.
- **Time shift**: We load in a piano roll of 2 times the desired length, and randomly select a starting time to obtain the actual piano roll for training.
- **Tempo shift**: Tempo of the piece was shifted by a factor of [0.95, 1.05].

VAE. Utilizing the standard autoencoder architecture from (Rombach et al., 2022), we compressed piano roll segments (dimension $3 \times 128 \times 128$) into a latent space

of $4 \times 16 \times 16$. The three dimensions of the piano roll include onset and pedal information, in addition to the standard piano roll data.

Let x represent the piano roll in pixel space and z the latent code. We denote the encoder and decoder as $\mathcal{E}$ and $\mathcal{D}$, respectively. The training objective for our VAE model is formulated as follows:

$$\begin{aligned}\mathcal{L}_{VAE} =& \|\mathcal{D}(\mathcal{E}(x)) - x\|_1 \\ &+ \lambda_{\text{KL}}(t) D_{KL}(\mathcal{N}(z; \mathcal{E}_\mu, \mathcal{E}_{\sigma^2}) \| \mathcal{N}(z; 0, I)) \\ &+ \lambda_{\text{denoise}}(t) \|\mathcal{D}(\mathcal{E}(\text{Noisy}(x))) - x\|_1 \,.\end{aligned} \tag{D.11}$$

The first term is the standard reconstruction loss, where we used L1 loss to encourage sparsity. The second term is the standard KL regularization term weighted by a scheduler $\lambda_{\text{KL}}(t)$. The third term is a denoising reconstruction loss, influenced by the scheduler $\lambda_{\text{denoise}}(t)$. Here, Noisy refers to a perturbation operator applied to the piano roll, encompassing:

- **Note shift**: Some fraction of notes were randomly selected by uniform distribution to be perturbed. Perturbed notes were shifted by up to 2 pitches higher or lower.
- **Adjacent note addition**: Some fraction of notes were randomly selected by uniform distribution. A second identical note was added just one pitch higher or lower to the original note. These adjacent notes are quite discordant to the ear.
- **Rhythm shift**: Some fraction of notes were randomly selected by uniform distribution to be perturbed. Perturbed notes were shifted by up to 100 ms earlier or later.
- **Note deletion**: Some fraction of notes were randomly selected by uniform distribution to be deleted.

We capped the maximum fraction of perturbed notes at 25% for all perturbations. The model was trained over 800k steps. The KL scheduler, $\lambda_{\text{KL}}(t)$, was a linear scheduler increasing from 0 to $1e-2$ across 400k steps. The denoising scheduler, λ_{denoise}, linearly increased the perturbation fraction from 0 to 25% over 400k steps. We employed a cosine learning rate scheduler with a 10k-step warmup, peaking at a

learning rate of $5e-4$. The optimizer used was AdamW (Loshchilov and Hutter, 2019), with weight decay of 0.01 and a batch size of 80.

Diffusion Model. We train our diffusion model with a transformer backbone on the latent space of a pretrained VAE. First we rescale the latent representation $\mathcal{E}(x)$ by its standard deviation, computed using a batch of 256 training samples as per the methodology described in (Rombach et al., 2022). Then we reshaped the latent representation from $4 \times 16 \times 128$ to 32×256, followed by a transformation of the 32-dimensional vector to match the hidden dimension of the transformer backbone. We employed the DiT-XL architecture from (Peebles and Xie, 2023), which has a hidden dimension of 1152. In addition, we use rotary positional embedding (Su et al., 2023) to accommodate for various length of input (e.g., when generating longer sequence of music).

Given our use of data augmentation during diffusion model training, it was necessary to compute $\mathcal{E}(x)$ dynamically, a process that is typically time-consuming. To optimize this, we employed a strategy to avoid encoding each sample from scratch. During data loading, we initially loaded a piano roll of length 2560 and then encoded each 128-length segment using the pretrained encoder, resulting in 20 latent codes. By concatenating subsets of these latent codes, we generated 4 training samples (segments 1-8, 5-12, 9-16, and 13-20), each measuring $8 \times 128 = 1024$ in length.

We train our model on three datasets using the training procedure of classifier-free guidance (Ho and Salimans, 2022). Specifically, we set $y = 0$ for Maestro, $y = 1$ for Muscore and $y = 2$ for Pop. We jointly train a conditional model $\epsilon_\theta(\mathbf{x}_t, t, y)$ and an unconditional model $\epsilon_\theta(\mathbf{x}_t, t, \text{null})$ with a dropout rate of 0.1.

We adhered to the training hyper-parameters outlined in (Peebles and Xie, 2023). Specifically, we used a constant learning rate of $1e-4$, no weight-decay and a batch size of 256 with the AdamW optimizer (Loshchilov and Hutter, 2019). We use linear noise scheduling and trained the model for 1.2M steps.

Objective Evaluation Setup

Unconditional Generation. In our study, we generated 400 music segments, each lasting 10.24 seconds, for all the methods under consideration. For baseline methods that utilize bars as the time unit, we produced 8-bar sequences from which we randomly extracted segments of 10.24 seconds in duration. We used the official released models for Remi (Huang and Yang, 2020), CPW (Hsiao et al., 2021) and PolyDiff (Min et al., 2023). Unfortunately, an official implementation for the music

transformer (Huang et al., 2018) was not available. Consequently, we resorted to an unofficial implementation[1] and trained a music transformer by ourselves.

The overlapping area (OA) for seven music attributes was computed following the methodology described in (Yang and Lerch, 2020). To accurately evaluate OA, it is typically required that the two datasets being compared contain an equal amount of data. Therefore, we randomly selected 400 samples from the test dataset to align with the number of generated samples. This evaluation process was repeated five times to calculate the mean and standard deviation of the results in Table 5.2.

Individual Rule Guidance. For each rule, we randomly selected 200 samples from the Muscore test dataset and computed their corresponding rule labels to serve as targets. Subsequently, we generated 200 samples conditioned on each rule label. The rule labels for these generated samples were computed, and the loss between the generated rule label and the target was calculated. The mean and standard deviation of this loss across the 200 samples are presented in Table 5.3.

Composite Rule Guidance. We randomly selected 200 samples from the Muscore test dataset and compute their rule labels for each of the three rules under consideration. Then we generated 200 samples conditioned on all three rule labels simultaneously, with the intention that the generated samples adhere to all three rules concurrently.

Ablation Studies. For all the ablation studies, we follow the individual rule guidance set up and guide the diffusion model to generate music following given note density. The computational time is for generating 4 samples in a batch. Regarding the quality metric, we randomly chose 200 samples from the test dataset and calculated the average OA similar to the process used for unconditional generation. This procedure was repeated five times to calculate the mean and standard deviation.

Editing. To facilitate a comparison with MuseMorphase (Wu and Yang, 2023), we adopted their approach for computing the note density label. Initially, we randomly selected 200 samples from the Pop test dataset and calculated both vertical and horizontal note density vectors for each sample, using a window size of 1.28 seconds. Consequently, for each 10.24-second sample, we obtained 8 vertical and 8 horizontal note density values. We then flattened these note density vectors and categorized them into 8 bins, ensuring approximately an equal number of samples in each bin for both vertical and horizontal densities. During generation, we randomly chose

[1]Available at `https://github.com/gwinndr/MusicTransformer-Pytorch`

a shift value of -1, 0, or 1 to adjust the note density classes of a sample, using the center value of the resultant bin as the target note density.

We also considered PolyDiff (Min et al., 2023) as another baseline. In their approach, a new musical piece is generated based on the piano roll with basic chords extracted from the current piece, which is seen as an editing task since it creates a variation of the existing music with the same chord progression. In our framework, we extracted chords from the source music, added noise to the source, and then generated new music guided by the extracted chords.

For both baseline methods, we generated 8-bar music segments and extracted rule labels for each bar. In contrast, our method involved generating 10.24-second music segments and using a 1.28-second time window to extract rule labels, thereby aligning the number of rule labels across all methods.

In terms of similarity metrics, we calculated chroma and grooving similarity between the generated samples and their respective source samples, following the methodology outlined in (Wu and Yang, 2023).

D.5 Training Surrogate Models for Music Rules

For classifier guidance (Dhariwal and Nichol, 2021) and DPS-NN (Chung et al., 2023), we need to train surrogate models to approximate various music rules. We used the DiT-S architecture in (Peebles and Xie, 2023) as the backbone for classifiers [2]. Following the ViT approach (Dosovitskiy et al., 2021), we appended a class token to the latent codes and utilized a multi-layer perceptron (MLP) for the classification head. For rules like pitch histogram and note density, our classifier produces a vector of corresponding rules, and we train it using L2 loss. For chord progression, we incorporated two classifier heads: one to predict the key logits and the other for chord logits for each 128-length segment. We treated key and chord as categorical variables and trained the model using cross-entropy loss.

Figure D.1 illustrates the training and validation loss/accuracy for the three rules under study. Notably, training a surrogate model for chord progression proved to be particularly challenging, with the final accuracy hovering around only 33%. This lower accuracy partly accounts for the diminished performance of rule-guided methods that depend on surrogate models.

[2]We use classifiers to refer to the surrogate models, even if they are not necessarily trained using a classification objective

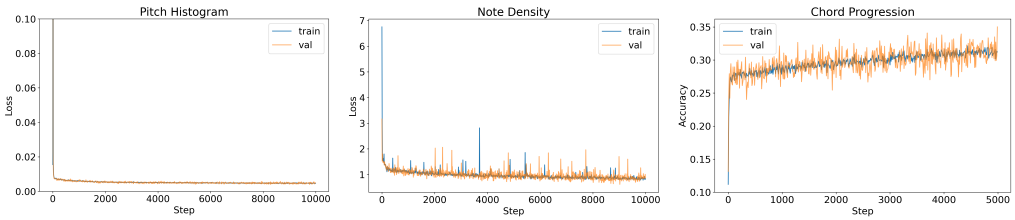


Figure D.1: **Training and validation curves of the classifiers trained on various rules.**

D.6 Losses over Stochastic Control Guided Sampling Process

We recorded the lowest loss and the variation in losses at each step throughout the sampling process of a representative sample using SCG, with note density as the conditioning rule. As depicted in Figure D.2 (a), we observed that the loss remains consistent until approximately $t = 750$. This early-stage constancy is attributed to the fact that, initially, the decoded piano rolls are essentially empty following the background thresholding, leading to a zero note density and, consequently, a stable loss. However, as the decoded piano rolls begin to populate, various realizations yield different note densities, resulting in a diversity of losses. By selecting the lowest loss at every step, we achieved a decrease in overall loss.

Figure D.2 (b) illustrates the range of the losses at each step. The largest range occurs around $t = 750$, the point where the piano roll starts to gain semantic meaning and the best loss drops drastically. This suggests that applying guidance early, soon after the piano roll acquires semantic content, is crucial for successful guidance.

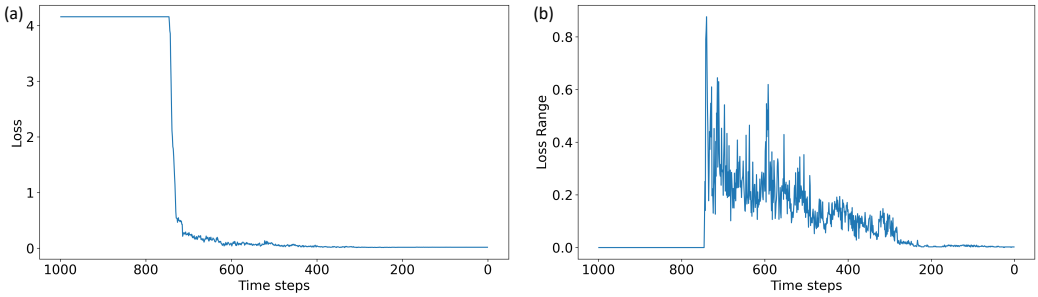


Figure D.2: **Best loss (a) and loss range (b) over stochastic control guided DDPM sampling on a representative sample with note density as the conditioning rule.**

D.7 More Ablation Studies

Unconditional Generation

Our model is capable of generating samples reflective of the distributions from three distinct datasets. This is accomplished through classifier-free guidance [3] (Ho and Salimans, 2022), with conditioning based on the specific dataset. We tuned the strength of the classifier-free guidance for each dataset and discovered optimal settings for achieving the highest music quality (Table D.3). Specifically, for the Maestro and Pop datasets, a guidance strength of $\omega = 0$ yielded the best results. In contrast, for the Muscore dataset, setting $\omega = 4$ proved to be most effective in enhancing musical quality.

Dataset	w	Used Pitch	IOI	Pitch Hist	Pitch Range	Velocity	Note Duration	Note Density	Avg
Maestro	0	$\mathbf{0.961 \pm 0.006}$	0.901 ± 0.009	0.960 ± 0.010	$\mathbf{0.963 \pm 0.006}$	$\mathbf{0.971 \pm 0.004}$	$\mathbf{0.910 \pm 0.012}$	0.934 ± 0.005	$\mathbf{0.943 \pm 0.003}$
	1	0.948 ± 0.002	$\mathbf{0.915 \pm 0.011}$	0.946 ± 0.004	0.931 ± 0.009	0.956 ± 0.008	$\mathbf{0.910 \pm 0.007}$	0.937 ± 0.008	0.935 ± 0.003
	2	0.953 ± 0.004	0.878 ± 0.006	$\mathbf{0.964 \pm 0.003}$	0.946 ± 0.006	0.963 ± 0.008	0.892 ± 0.009	$\mathbf{0.953 \pm 0.004}$	0.935 ± 0.001
	4	0.925 ± 0.005	0.891 ± 0.008	0.940 ± 0.007	0.934 ± 0.011	0.958 ± 0.010	0.890 ± 0.010	0.932 ± 0.005	0.924 ± 0.001
Muscore	0	0.941 ± 0.003	0.890 ± 0.019	0.950 ± 0.014	0.946 ± 0.006	0.886 ± 0.010	$\mathbf{0.922 \pm 0.011}$	0.925 ± 0.006	0.923 ± 0.003
	1	0.956 ± 0.008	0.870 ± 0.014	0.942 ± 0.007	0.962 ± 0.009	0.885 ± 0.006	0.911 ± 0.007	0.934 ± 0.007	0.923 ± 0.003
	2	0.942 ± 0.009	0.899 ± 0.014	0.940 ± 0.014	0.954 ± 0.008	0.884 ± 0.014	0.911 ± 0.007	0.924 ± 0.008	0.922 ± 0.005
	4	$\mathbf{0.963 \pm 0.004}$	$\mathbf{0.915 \pm 0.009}$	$\mathbf{0.962 \pm 0.009}$	$\mathbf{0.964 \pm 0.004}$	$\mathbf{0.890 \pm 0.006}$	0.900 ± 0.012	$\mathbf{0.946 \pm 0.005}$	$\mathbf{0.934 \pm 0.003}$
Pop	0	0.927 ± 0.009	$\mathbf{0.952 \pm 0.004}$	0.969 ± 0.002	0.928 ± 0.013	$\mathbf{0.948 \pm 0.003}$	0.911 ± 0.019	0.941 ± 0.009	$\mathbf{0.939 \pm 0.004}$
	1	0.921 ± 0.004	0.917 ± 0.003	$\mathbf{0.975 \pm 0.003}$	$\mathbf{0.937 \pm 0.009}$	0.939 ± 0.007	$\mathbf{0.926 \pm 0.015}$	0.935 ± 0.005	0.936 ± 0.003
	2	0.929 ± 0.008	0.885 ± 0.010	0.970 ± 0.005	0.926 ± 0.013	0.935 ± 0.007	$\mathbf{0.926 \pm 0.012}$	0.950 ± 0.011	0.932 ± 0.003
	4	$\mathbf{0.933 \pm 0.011}$	0.897 ± 0.004	0.965 ± 0.007	0.920 ± 0.010	0.940 ± 0.007	0.914 ± 0.015	$\mathbf{0.962 \pm 0.007}$	0.933 ± 0.005

Table D.3: Unconditional generation on three datasets with different classifier-free guidance strength.

Latent vs Pixel Space

Our approach employed a latent diffusion model for symbolic music generation and compared its performance with a diffusion model trained in pixel space. The pixel space model was configured with a time resolution of 0.08 seconds per column in the piano roll, as opposed to the 0.01-second resolution in latent space. This choice was primarily driven by computational constraints; a 0.01-second resolution for a 10.24-second music piece would result in a piano roll of size $3 \times 128 \times 1024$, posing significant computational demands. In contrast, a 0.08-second resolution yields a more manageable size of $3 \times 128 \times 128$. For training the pixel space diffusion model, we utilized a standard U-Net backbone.

Table D.4 presents a comparison of the models in the task of unconditional generation. An intriguing trend emerged: the latent space model excelled with complex, dynamic-rich music (e.g., Maestro), whereas the pixel space model showed superior performance with simpler music (e.g., Pop). The Muscore dataset, predominantly

[3] Despite this approach, we refer to it as 'unconditional generation' because it does not involve rule-based guidance.

featuring classical sheet music, sits between Maestro and Pop in terms of complexity, and here, both models performed comparably. This observation aligns with the notion that time resolution has a less pronounced impact on the expressiveness of simpler music, making a lower resolution viable for training the diffusion model.

dataset	method	Used Pitch	IOI	Pitch Hist	Pitch Range	Velocity	Note Duration	Note Density	Avg
Maestro	pixel	0.919 ± 0.005	0.877 ± 0.018	$\mathbf{0.983 \pm 0.005}$	0.959 ± 0.007	0.969 ± 0.003	0.897 ± 0.013	0.896 ± 0.003	0.929 ± 0.006
	latent	$\mathbf{0.961 \pm 0.006}$	$\mathbf{0.901 \pm 0.009}$	0.960 ± 0.010	$\mathbf{0.963 \pm 0.006}$	$\mathbf{0.971 \pm 0.004}$	$\mathbf{0.910 \pm 0.012}$	$\mathbf{0.934 \pm 0.005}$	$\mathbf{0.943 \pm 0.003}$
Muscore	pixel	0.962 ± 0.005	0.903 ± 0.009	$\mathbf{0.965 \pm 0.009}$	$\mathbf{0.964 \pm 0.007}$	$\mathbf{0.893 \pm 0.007}$	$\mathbf{0.928 \pm 0.011}$	0.926 ± 0.008	$\mathbf{0.934 \pm 0.005}$
	latent	$\mathbf{0.963 \pm 0.004}$	$\mathbf{0.915 \pm 0.009}$	0.962 ± 0.009	$\mathbf{0.964 \pm 0.004}$	0.890 ± 0.006	0.900 ± 0.012	$\mathbf{0.946 \pm 0.005}$	$\mathbf{0.934 \pm 0.003}$
Pop	pixel	$\mathbf{0.935 \pm 0.011}$	$\mathbf{0.957 \pm 0.004}$	$\mathbf{0.976 \pm 0.003}$	$\mathbf{0.952 \pm 0.004}$	0.945 ± 0.006	$\mathbf{0.935 \pm 0.011}$	$\mathbf{0.946 \pm 0.013}$	$\mathbf{0.949 \pm 0.005}$
	latent	0.927 ± 0.009	0.952 ± 0.004	0.969 ± 0.002	0.928 ± 0.013	$\mathbf{0.948 \pm 0.003}$	0.911 ± 0.019	0.941 ± 0.009	0.939 ± 0.004

Table D.4: Comparing pixel vs latent space for unconditional generation.

Table D.5 shows the loss for individual rule guidance using the pixel space-trained diffusion model. Mirroring the findings in Table 5.3, SCG consistently achieved the lowest loss, underscoring its effectiveness in rule guidance. However, a noticeable decline in music quality (measured by OA) was observed for the model trained on pixel space, particularly in aspects like pitch histogram and note density (Table D.6). This decline can be attributed to the nature of Gaussian noise addition in pixel space, which often results in random, musically nonsensical notes that nevertheless align with rule targets. Conversely, noise addition in latent space tends to induce more meaningful alterations, thereby preserving the higher music quality.

Method	Pitch Histogram ↓	Note Density ↓	Chord Progression ↓
No Guidance	0.019 ± 0.011	2.367 ± 2.933	0.841 ± 0.142
Classifier	0.020 ± 0.015	0.287 ± 0.330	0.783 ± 0.208
DPS - NN	0.020 ± 0.013	0.615 ± 1.188	0.788 ± 0.170
DPS - Rule	0.002 ± 0.006	2.349 ± 3.425	-
SCG	$\mathbf{0.0001 \pm 0.0008}$	$\mathbf{0.103 \pm 0.570}$	$\mathbf{0.344 \pm 0.212}$

Table D.5: Loss between the target and the generated attributes for individual rule guidance using the pixel-space trained diffusion model.

Model	Pitch Histogram ↑	Note Density ↑	Chord Progression ↑
Pixel	0.848 ± 0.005	0.797 ± 0.005	$\mathbf{0.892 \pm 0.009}$
Latent	$\mathbf{0.897 \pm 0.006}$	$\mathbf{0.880 \pm 0.003}$	0.883 ± 0.002

Table D.6: Comparison of Average Overlapping Area (OA) for individual rule guidance between diffusion models trained on pixel and latent space.

Composite Rule Guidance

In the task of composite rule guidance, the allocation of suitable weights to each rule is crucial for effective rule-based guidance. Table D.7 shows the performance associated with various weight assignments. Generally, we observed that amplifying

the weight assigned to a specific rule tends to decrease the loss pertinent to that rule. However, excessively concentrating the weight on a single rule can lead to a deterioration in performance, as evidenced by the configuration with a 40-1-4 weight assignment with an overly heavy emphasis on chord progression (CP).

Weight	PH ↓	ND ↓	CP ↓	Quality ↑
40-1-1	0.004 ± 0.005	0.218 ± 0.243	0.447 ± 0.226	0.901 ± 0.003
40-1-2	0.004 ± 0.004	$\mathbf{0.215 \pm 0.193}$	$\mathbf{0.392 \pm 0.206}$	$\mathbf{0.905 \pm 0.007}$
40-1-4	0.004 ± 0.004	0.251 ± 0.236	0.418 ± 0.216	0.884 ± 0.011
100-1-1	$\mathbf{0.003 \pm 0.002}$	0.236 ± 0.244	0.434 ± 0.229	0.903 ± 0.005

Table D.7: Composite rule guidance using Classifier + SCG-4 with different weight on each rule. The weight column displays the weight in the order of PH, ND and CP.

Additionally, we investigated the impact of the sample count n on composite rule guidance, as shown in Table D.8. The observed trend is consistent with that in individual rule guidance: using a greater number of samples at each step results in a lower loss. Another noteworthy observation is that combining SCG with other guidance methods (e.g., classifier guidance) and using a smaller sample count n (such as 4) can yield better outcomes than using SCG alone with $n = 16$. This improvement occurs because classifier guidance provides a coarse guidance signal, making it easier to identify advantageous directions based on these preliminary signals. As expected, the hybrid approach with a larger sample count $n = 16$ achieves the lowest loss. Remarkably, the losses in this case are similar to those in individual rule guidance, despite being achieved simultaneously.

Method	n	PH ↓	ND ↓	CP ↓	Quality ↑
SCG	16	0.014 ± 0.009	0.466 ± 0.648	0.446 ± 0.205	$\mathbf{0.909 \pm 0.005}$
DPS-NN + SCG	4	0.003 ± 0.004	0.392 ± 0.612	0.486 ± 0.270	0.826 ± 0.005
Classifier + SCG	4	0.004 ± 0.005	0.218 ± 0.243	0.447 ± 0.226	0.901 ± 0.003
DPS-NN + SCG	16	$\mathbf{0.002 \pm 0.007}$	0.238 ± 0.531	0.313 ± 0.231	0.844 ± 0.007
Classifier + SCG	16	0.003 ± 0.005	$\mathbf{0.148 \pm 0.203}$	$\mathbf{0.284 \pm 0.197}$	0.894 ± 0.007

Table D.8: Effect of number of samples n on composite rule guidance.

D.8 Rule-Guided Generation Survey

Details

To evaluate the rule alignment of our approach SCG compared to two baseline methods, we designed a listening test. We extracted three musical rules (Pitch Histogram, Note Density, and Chord Progression) from various segments in our dataset. Next, we created music samples lasting 10.24 seconds each, adhering to these three rules. We produced four samples for each guiding method, including our own, resulting in a total of 12 samples.

We recruited 15 participants with substantial musical experience for our survey. We gathered information on their musical backgrounds, including the number of years they have been playing music, their years of formal music education, and the instruments they play. A significant portion of the participants have over 10 years of experience in both playing music and formal musical study, as shown in figures D.3 and D.4. Figure D.5 displays a diverse range of instrument expertise among participants, with a notable prevalence of piano players, aligning with our model's focus on piano music. This diversity and level of experience make the participants well-suited for analyzing the music's rule alignment and musicality.

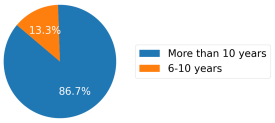


Figure D.3: Years of playing music

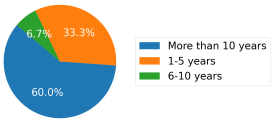


Figure D.4: Years of formal music study

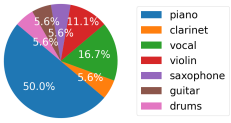


Figure D.5: Instruments of participants

The dimension we evaluate about the sample quality are rule alignment, creativity, coherence and overall rating. Question 1-3 are about rule alignment, which evaluates the performance of guidance. Creativity refers to whether samples are musically interesting or not. For example, if one segment is static, then such sample is not creative. Coherence refers to whether the samples align with basic musical common knowledge. For instance, if one segment contains many random notes, then such sample sounds chaotic and it is not coherent to human's sense of good music. The overall rating is evaluated by participants, where they give a score solely based on their preference to the samples.

Each evaluation dimension is rated on a scale up to 5 points. For rule alignment, a Likert scale is used, where "completely unaligned" is rated as 1 and "perfectly aligned" as 5. The average score from questions 1-3 determines the rule alignment score. Creativity is initially scored out of 3 points, which is then normalized to a 5-point scale. The average of questions 4 and 5 calculates the creativity score. In question 4, "No" scores 1 point, "Maybe" 2 points, and "Yes" 3 points. For question 5, both "Too Simple" and "Too Complex" score 1 point, while "Moderate" scores 3 points.

For coherence, the maximum score is 4 points, later normalized to 5 points. The average from questions 6-8 gives the coherence score. In question 6, "Many" errors score 1 point, "Some" 2 points, "A Few" 3 points, and "None" 4 points. In question 7,

"Mainly incoherent harmonic motion" scores 1 point, "Mainly incoherent harmonic motion with some coherence" 2 points, "Mainly coherent harmonic motion with some incoherence" 3 points, and "Coherent harmonic motion" 4 points. Question 8 uses a tailored scoring system to fit the 4-point scale: "Poor" is valued at 4/3 points, "Moderate" at 8/3 points, and "Highly Engaging" at 4 points.

The overall rating also utilizes a Likert scale, with "Poor" equating to 1 point, "Fair" 2 points, "Good" 3 points, "Very Good" 4 points, and "Excellent" 5 points.

Regarding the music rules (Pitch Histogram, Note Density, and Chord Progression), we generated the entire sample conditioned on pitch histogram. To assess note density and chord progression, the music segment is divided into 8 equal-length segments or windows. We then analyze and compute the note density and chord progression within each of these windows. The survey consists 9 questions in total, investigating SCG's rule alignment and sample's musicality.

Survey

The first three questions of the survey are studying the rule alignment of guidance mechanisms. The answer of these three questions are all classified into 5 categories instead of binary choices because rule alignment can be effective for parts of the music sample. For example, given a 10 second music sample, the first 5 seconds of the sample has the perfect alignment, and the last 5 seconds of the sample does not align with provided rules at all. In this case, only binary classification on how effective the guidance is would not be enough to distinguish such sample. Thus, we construct 5 options instead of binary choices.

Question 1: On a scale of 1 to 5, how well does the pitch histogram in the sample music match the provided histogram? (1 indicating the least alignment, with 5 indicating the most alignment)

The options are:

- Completely unaligned (1): The pitch histogram in the sample music is completely different from the provided pitch histogram.
- Somewhat unaligned (2): The pitch histogram in the sample music is somewhat different from the provided pitch histogram, with a small portion of the segment aligned.

- Moderately aligned (3): The pitch histogram in the sample music is somewhat aligned with the provided pitch histogram, with a small portion of the segment not aligned.
- Mostly aligned (4): The pitch histogram in the sample music is mostly aligned with the provided pitch histogram.
- Perfectly aligned (5): The pitch histogram in the sample music is perfectly aligned with the pitch histogram.

Besides the generated, we show the participants the image of pitch histogram that are used for guidance. Note that Pitch histogram is the distribution of notes. Question 1 focuses on the alignment of pitch histogram, which means whether distribution of notes in the given sample follows the given pitch histogram. The question directly evaluates how effective the conditioning on pitch histogram is.

Question 2: On a scale of 1 to 5, how would you rate the alignment of the note density of the sample music compared to the note density provided in the above youtube video? (1 being the least aligned, 5 being the most aligned, take a look at the piano roll image would be a good idea)

The options are:

- Completely unaligned (1): The note density in the sample music significantly differs from the provided music segment, leading to a large disparity in musical texture and pacing.
- Somewhat unaligned (2): The note density in the sample music somewhat differs from the provided music segment, causing a noticeable disparity in musical texture and pacing.
- Moderately aligned (3): The note density in the sample music is somewhat aligned with the provided note density, but with a perceptible mismatch in musical flow and rhythm.
- Mostly aligned (4): The note density of the sample music closely matches the provided note density, with slight differences in how notes are spaced and arranged.

- Perfectly aligned (5): The note density of the sample music aligns perfectly with the provided note density, reflecting a very similar density pattern in the distribution of notes.

Question 2 evaluates the alignment of note density. Note density refers to the frequency and distribution of musical notes in a piece, indicating how many notes occur over a specific time or within a certain section of the music. In other words, note density reflects the texture and pacing in music segments. Thus, under a successful guidance of such rule, the texture and pacing of generated samples would be similar to the density pattern of corresponding segments in the distribution of notes.

Question 3: On a scale from 1 to 5, how well does the chord progression in the sample music match the provided chord progression, focusing on their functional harmony and general sequence rather than specific chord inversions. (1 indicates minimal alignment, with pronounced differences in chord progression, 5 signifies complete alignment, with the chord progressions being very similar or identical)

The options are:

- (Completely Unaligned): The bass line of the chord progression in the sample music significantly deviates from the provided progression, resulting in a clear disparity in harmonic structure and musical direction.
- (Somewhat Unaligned): Observable differences in the chord progression between the sample music and the provided example lead to a discordant sound and a disrupted musical flow.
- (Moderately Aligned): The chord progression in the sample music is somewhat consistent with the provided progression, with only minor discrepancies in the sequence or harmony.
- (Mostly Aligned): The chord progression in the sample closely mirrors the provided progression, with only negligible variations that don't substantially affect the overall harmonic continuity.
- (Perfectly Aligned): The chord progression in the sample music perfectly matches the provided progression, ensuring a cohesive and harmonious harmonic structure throughout.

Chord progression guides the music segment sounding more reasonable. Such questions asks about the alignment of chords, where effectively evaluates the controllable generation based on given chord progression.

The subsequent five questions explore the musicality of the generated samples, encompassing both creativity and coherence. The primary aim of rule-based guidance is to enhance the auditory appeal of the samples, making them more enjoyable to listeners. A generation is not considered successful if it fails to be aesthetically pleasing, regardless of achieving perfect alignment with all three specified rules.

Questions 4 and 5 focus on evaluating the creativity of the music sample.

Question 4: Do you like the music based on your personal taste?

The options are:

- Yes
- No
- Maybe

Questions 4 directly asks whether the participants like the music based on their personal taste. The evaluation from listeners with substantial musical experience illustrates the quality of model generation.

Question 5: What do you think of the complexity of this music?

The options are:

- Too Simple
- Moderate
- Too Complex

Question 5 assesses the complexity of the music, indicating that a moderate level of complexity is optimal. Music that is either too simple or too complex is considered to detract from the quality of the sample.

Questions 6 to 9 are designed to assess the coherence of the music sample.

Question 6: How many elements in the sample that seem out of place or random?

The options are:

- None
- A Few
- Some
- Many

Question 6 aims to evaluate on the generation quality of the model. Because the sample is composed by the model instead of human, such sample might have random notes. The random elements would break the entity of the music segment and the pleasure of listening for participants.

Question 7: How coherent do you find the harmony in the excerpt to be?

The options are:

- Coherent harmonic motion
- Mainly coherent harmonic motion with some incoherence
- Mainly incoherent harmonic motion with some coherence
- Mainly incoherent harmonic motion

Question 7 assesses the harmonic coherence of the generated sample, specifically evaluating if the music segment appears harmonically random or structured.

Question 8: How would you rate the appropriateness and engagement of the texture in the sample music, considering the layers and how they combine?

The options are:

- Highly Engaging
- Moderate
- Poor

Question 8 examines the texture of the sample music, focusing on whether the music presents an overly complex or disordered structure.

Question 9 solicits the overall rating of the music, where participants rate the sample according to their personal preferences.

Question 9: Overall Rating: On a scale of 1 to 5, how would you rate this sample? (1 being lowest, 5 highest)

The options are:

- Poor: The music lacks appeal in many aspects and does not engage the listener.
- Fair: The music has some redeeming qualities but falls short in several areas.
- Good: The music is enjoyable and well-composed, though it may have a few minor flaws.
- Very Good: The music is engaging and impressive, showing high levels of creativity and skill.
- Excellent: The music is outstanding in all respects, offering a deeply satisfying and memorable listening experience.

References

Jooyoung Choi, Sungwon Kim, Yonghyun Jeong, Youngjune Gwon, and Sungroh Yoon (2021). "Ilvr: Conditioning Method for Denoising Diffusion Probabilistic Models". In: *International Conference on Computer Vision*.

Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye (2023). "Diffusion Posterior Sampling for General Noisy Inverse Problems". In: *International Conference on Learning Representations*.

Michael Scott Cuthbert and Christopher Ariza (2010). "Music21: A Toolkit for Computer-aided Musicology and Symbolic Music Data". In: *International Society for Music Information Retrieval*.

Paolo Dai Pra (1991). "A stochastic control approach to reciprocal diffusion processes". In: *Applied mathematics and Optimization* 23, pp. 313–329.

Prafulla Dhariwal and Alexander Nichol (2021). "Diffusion Models Beat GANs on Image Synthesis". In: *Advances in Neural Information Processing Systems* 34, pp. 8780–8794.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*.

Jonathan Ho and Tim Salimans (2022). "Classifier-free Diffusion Guidance". In: *arXiv preprint arXiv:2207.12598*.

Wen-Yi Hsiao, Jen-Yu Liu, Yin-Cheng Yeh, and Yi-Hsuan Yang (2021). "Compound Word Transformer: Learning to Compose Full-song Music Over Dynamic Directed Hypergraphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 1, pp. 178–186.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck (2018). "Music Transformer". In: *arXiv preprint arXiv:1809.04281*.

Yu-Siang Huang and Yi-Hsuan Yang (2020). "Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions". In: *Proceedings of the 28th ACM international conference on multimedia*, pp. 1180–1188.

Bahjat Kawar, Michael Elad, Stefano Ermon, and Jiaming Song (2022). "Denoising Diffusion Restoration Models". In: *Advances in Neural Information Processing Systems* 35, pp. 23593–23606.

Ilya Loshchilov and Frank Hutter (2019). "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*.

Chenlin Meng, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon (2022). "SDEdit: Image Synthesis and Editing with Stochastic Differential Equations". In: *International Conference on Learning Representations*.

Lejun Min, Junyan Jiang, Gus Xia, and Jingwei Zhao (2023). "Polyffusion: A Diffusion Model for Polyphonic Score Generation with Internal and External Controls". In: *Proc. of the 24th Int. Society for Music Information Retrieval Conf.*

Michele Pavon (1989). "Stochastic Control and Nonequilibrium Thermodynamical Systems". In: *Applied Mathematics and Optimization* 19, pp. 187–202.

William Peebles and Saining Xie (2023). "Scalable Diffusion Models with Transformers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4195–4205.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer (2022). "High-resolution Image Synthesis with Latent Diffusion Models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695.

Jiaming Song, Chenlin Meng, and Stefano Ermon (2021). "Denoising Diffusion Implicit Models". In: *International Conference on Learning Representations*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu (2023). "Roformer: Enhanced Transformer with Rotary Position Embedding". In: *Neurocomputing*, p. 127063.

Shih-Lun Wu and Yi-Hsuan Yang (2023). "MuseMorphose: Full-song and Fine-grained Piano Music Style Transfer with One Transformer VAE". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 31, pp. 1953–1967.

Li-Chia Yang and Alexander Lerch (2020). “On the Evaluation of Generative Models in Music”. In: *Neural Computing and Applications* 32.9, pp. 4773–4784.