On the Forward Invariance of Neural ODEs

Wei Xiao [1] **Tsun-Hsuan Wang** [1] **Ramin Hasani** [1] **Mathias Lechner** [1]
Yutong Ban [1] **Chuang Gan** [2] **Daniela Rus** [1]

Abstract

We propose a new method to ensure neural ordinary differential equations (ODEs) satisfy output specifications by using invariance set propagation. Our approach uses a class of control barrier functions to transform output specifications into constraints on the parameters and inputs of the learning system. This setup allows us to achieve output specification guarantees simply by changing the constrained parameters/inputs both during training and inference. Moreover, we demonstrate that our invariance set propagation through data-controlled neural ODEs not only maintains generalization performance but also creates an additional degree of robustness by enabling causal manipulation of the system's parameters/inputs. We test our method on a series of representation learning tasks, including modeling physical dynamics and convexity portraits, as well as safe collision avoidance for autonomous vehicles.

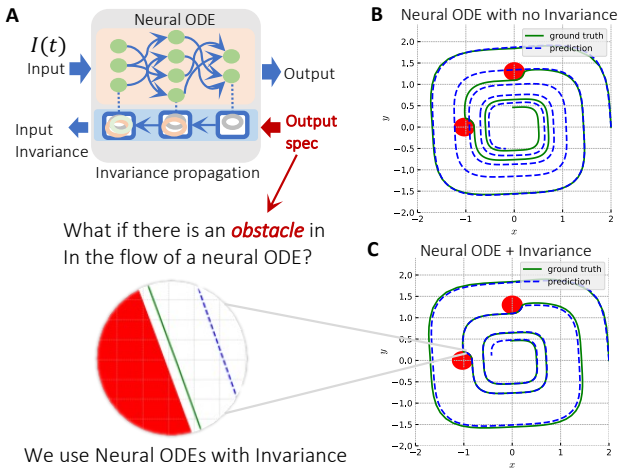

Figure 1. Invariance Propagation for neural ODEs. Output specifications can be guaranteed with invariance, including specification satisfaction between samplings, e.g., spiral curve regression with critical region avoidance.

1. Introduction

Neural ODEs (Chen et al., 2018) are continuous deep learning models that enable a range of useful properties such as exploiting dynamical systems as an effective learning class (Haber & Ruthotto, 2017; Gu et al., 2021), efficient time series modeling (Rubanova et al., 2019; Lechner & Hasani, 2022), and tractable generative modeling (Grathwohl et al., 2018; Liebenwein et al., 2021).

Neural ODEs are typically trained via empirical risk minimization (Rumelhart et al., 1986; Pontryagin, 2018) endowed with proper regularization schemes (Massaroli et al., 2020) without much control over the behavior of the obtained network and over the ability to account for counterfactual inputs (Vorbach et al., 2021). For example, a well-trained neural ODE instance that learned to chase a spiral dynamic (Fig. 1**B**), would not be able to avoid an object on its flow, even if it has seen this type of *output specification/constraint* during training. This shortcoming demands a fundamental fix to ensure the safe operation of these models specifically in safety-critical applications such as robust and trustworthy policy learning, safe robot control, and system verification (Lechner et al., 2020; Kim et al., 2021; Hasani et al., 2022).

In this paper, we set out to ensure neural ODEs satisfy output specifications. To this end, we introduce the concept of propagating invariance sets. An invariance set is a form of specification consisting of physical laws, mathematical expressions, safety constraints, and other prior knowledge of the structure of the learning task. We can ensure that neural ODEs are invariant to noise and affine transformations such as rotating, translating, or scaling an input, as well as to other uncertainties in training and inference.

To propagate invariance sets through neural ODEs we can use Lyapunov-based methods with forward invariance properties such as a class of control barrier functions (CBFs) (Ames et al., 2017), to formally guarantee that the output

[1]Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA. [2]MIT-IBM Watson AI Lab. Videos and code are available on the website: https://weixy21.github.io/invariance/. Correspondence to: Wei Xiao <weixy@mit.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

specifications are ensured. In order to account for non-linearity of the model, high-order CBFs (Xiao & Belta, 2019), a general form of CBFs, are required since high-relative degree constraints are introduced in such cases. CBFs perform this via migrating output specifications to the learning system's parameters or its inputs such that we can solve the constraints via forward calls to the learning system equipped with a quadratic program (QP). However, doing this requires a series of non-trivial novelties which we address in this paper. 1. CBFs are model-based Lyapunov methods, thus, they can only be used with data and systems with known dynamics. Here, we extend their formalism to work with unknown dynamics by the properties of neural ODEs. 2. CBFs are typically applied to systems with affine transformations. For neural ODEs with nonlinear activations, the propagation of invariance sets becomes a challenge. We fix this by incorporating a virtually linear space within the neural ODE to find simple parameter/input constraints.

Going back to Fig. 1**C**, we observe that by applying our forward invariance propagation method, we can correct the model and force the system trajectories to stay away from the red obstacles while maintaining the path of the ground truth spiral curve.

In summary, we make the following **new contributions**:

- We incorporate formal guarantees into neural ODEs via invariance set propagation.
- We use the class of Higher-order CBFs (Xiao & Belta, 2022) to propagate the invariance set in neural ODEs while addressing their challenges such as handling unknown dynamics and the nonlinearity of the neural ODEs as well as connecting the order concept in HOCBFs to that of network depth in neural ODEs.
- We demonstrate the effectiveness of our method on a variety of learning tasks and output specifications, including the modeling of physical systems and the safety of neural controllers for autonomous vehicles.

2. Preliminaries

In this section, we provide background on neural ODEs and forward invariance in control theory.

2.1. Neural ODEs

A neural ordinary differential equation (ODE) is defined in the form (Chen et al., 2018):

$$\dot{\boldsymbol{x}}(t) = f_\theta(\boldsymbol{x}(t)), \tag{1}$$

where $n \in \mathbb{N}$ is state dimension, $\boldsymbol{x} \in \mathbb{R}^n$ is the state and $\dot{\boldsymbol{x}}$ denotes the time derivative of $\boldsymbol{x}$, $f_\theta : \mathbb{R}^n \to \mathbb{R}^n$ is a neural network model parameterized by θ. The output of the neural ODE is the integral solution of (1). It can also take in external input, where the model is defined as:

$$\dot{\boldsymbol{x}}(t) = f'_\theta(\boldsymbol{x}(t), \mathbf{I}(t)), \tag{2}$$

where $n_{\mathbf{I}} \in \mathbb{N}$ is external input dimension, $\mathbf{I}(t) \in \mathbb{R}^{n_{\mathbf{I}}}$, $f'_\theta : \mathbb{R}^n \times \mathbb{R}^{n_{\mathbf{I}}} \to \mathbb{R}^n$ is a neural network model parameterized by θ. For notation convenience, we write as,

$$f_\theta = f_{\theta_{K,K+1}} \circ \cdots \circ f_{\theta_{1,2}} \tag{3}$$

where K is the number of layers, $f_{\theta_{k,k+1}}, k \in [1, K]$ is the forward process of the k'th layer, and we denote $\boldsymbol{z}_k = (f_{\theta_{k,k+1}} \circ \cdots \circ f_{\theta_{1,2}})(\cdot) \in \mathbb{R}^{n_k}$ the intermediate representation at the k'th layer and $\boldsymbol{z}_K = \dot{\boldsymbol{x}}$ is the output. $n_k \in \mathbb{N}$ denotes the number of neurons at layer k and $n_K = n$.

2.2. Forward Invariance in Control Theory

Consider an affine control system of the form

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u} \tag{4}$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^{n\times q}$ are locally Lipschitz, and $\boldsymbol{u} \in U \subset \mathbb{R}^q$, where U denotes a control constraint set.

Definition 2.1. (**Set invariance**): A set $C \subset \mathbb{R}^n$ is forward invariant for system (4) if its solutions for some $\boldsymbol{u} \in U$ starting at any $\boldsymbol{x}(t_0) \in C$ satisfy $\boldsymbol{x}(t) \in C, \forall t \geq t_0$.

Definition 2.2. (**Relative degree**): The relative degree of a differentiable function $b : \mathbb{R}^n \to \mathbb{R}$ (or constraint $b(\boldsymbol{x}) \geq 0$) with respect to the system (4) is the number of times $b(\boldsymbol{x})$ needs to be differentiated along dynamics (4) until any component of $\boldsymbol{u}$ explicitly shows in the corresponding derivative.

Definition 2.3. (**Class $\mathcal{K}$ function**): A Lipschitz continuous function $\alpha : [0, a) \to [0, \infty), a > 0$ belongs to class $\mathcal{K}$ if it is strictly increasing and $\alpha(0) = 0$.

Definition 2.4. (**High Order Barrier Function (HOBF)**): A function $b : \mathbb{R}^n \to \mathbb{R}$ of relative degree m is a HOBF with a sequence of functions $\psi_i : \mathbb{R}^n \to \mathbb{R}$ such that $\psi_m(\boldsymbol{x}) \geq 0$,

$$\psi_i(\boldsymbol{x}) := \dot{\psi}_{i-1}(\boldsymbol{x}) + \alpha_i(\psi_{i-1}(\boldsymbol{x})), \quad i \in \{1, \ldots, m\}, \tag{5}$$

where $\psi_0(\boldsymbol{x}) := b(\boldsymbol{x})$ and $\alpha_i(\cdot)$ is a $(m-i)$'th order differentiable class $\mathcal{K}$ function. We define a sequence of sets,

$$C_i := \{\boldsymbol{x} \in \mathbb{R}^n : \psi_{i-1}(\boldsymbol{x}) \geq 0\}, \quad i \in \{1, \ldots, m\}. \tag{6}$$

Definition 2.5. (**High Order Control Barrier Function (HOCBF)** (Xiao & Belta, 2022)): Let ψ_i and C_i be defined by (5) and (6), respectively, for $i \in \{1, \ldots, m\}$. A function $b : \mathbb{R}^n \to \mathbb{R}$ is a HOCBF of relative degree m if there exists

$(m-i)^{th}$ order differentiable class $\mathcal{K}$ functions $\alpha_i, i \in \{1,\dots,m\}$ such that

$$\sup_{\boldsymbol{u}\in U}[L_f^m b(\boldsymbol{x}) + [L_g L_f^{m-1} b(\boldsymbol{x})]\boldsymbol{u} + O(b(\boldsymbol{x})) + \alpha_m(\psi_{m-1}(\boldsymbol{x}))] \geq 0, \tag{7}$$

for all $\boldsymbol{x} \in C_1 \cap, \dots, \cap C_m$. L_f and L_g denote Lie derivatives w.r.t. $\boldsymbol{x}$ along f and g, respectively, and $O(b(\boldsymbol{x})) = \sum_{i=1}^{m-1} L_f^i(\alpha_{m-i} \circ \psi_{m-i-1})(\boldsymbol{x})$. The satisfaction of (7) is equivalent to the satisfaction of $\psi_m(\boldsymbol{x}) \geq 0$ defined in (5).

The HOCBF is a general form of the CBF (Ames et al., 2017) (a HOCBF with $m = 1$ degenerates to a CBF), and it can be applied to arbitrary relative degree systems, such as the invariance propagation to nonlinear layers of a neural ODE in this work.

Theorem 2.6. *(Xiao & Belta, 2022)): Given a HOCBF $b(\boldsymbol{x})$ from Def. 2.5 with the sets $C_1, \dots, C_m$ defined by (6), if $\boldsymbol{x}(t_0) \in C_1 \cap, \dots, \cap C_m$, then any Lipschitz continuous controller $\boldsymbol{u}(t)$ that satisfies the constraint in (7), $\forall t \geq t_0$ renders $C_1 \cap, \dots, \cap C_m$ forward invariant for system (4).*

In this work, we map the forward invariance in control theory to *forward invariance in neural ODEs*, where we tackle arbitrary dynamics defined by neural ODE f_θ (which can be nonlinear as opposed to an affine control system in the form of (4)).

3. Invariance Propagation

In this section, we present the theoretical framework of *Invariance Propagation (IP)* to guarantee forward invariance (in short, invariance) of a neural ODE. We first provide formalisms of the proposed method. Then, we describe invariance propagation to (i) linear layer (ii) nonlinear layer (iii) external input.

Output Specification. A continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$ constructs an output specification $h(\boldsymbol{x}) \geq 0$ for a neural ODE. Typical output specifications include system safety (e.g., collision avoidance in autonomous driving), physical laws (e.g., energy conservation), mathematical formulae (e.g., Cauchy Schwarz inequality), etc.

Definition 3.1. (**Forward Invariance in Neural ODE**): The (forward) invariance of a neural ODE (1) or (2) with f_θ is defined w.r.t. its output specification $h(\boldsymbol{x}) \geq 0$ such that if $h(\boldsymbol{x}(t_0)) \geq 0$, then $h(\boldsymbol{x}(t)) \geq 0, \forall t \geq t_0$, where $x(t) = \int_{t_0}^{t} f_\theta(\tau) d\tau$. Intuitively, this property guarantees the satisfaction of output specification is forwarded in the neural ODE across time.

Definition 3.2. (**Invariance Propagation** (IP)): Given an output specification $h(\boldsymbol{x}) \geq 0$ and a neural ODE f_θ, invariance propagation describes a procedure to find a constraint $\Psi(\boldsymbol{d}) \geq 0$, where $\Psi : \mathbb{R}^{n_{\boldsymbol{d}}} \to \mathbb{R}$ and $\boldsymbol{d}$ ($n_d \in \mathbb{N}$ is its dimension) is either (i) a subset of parameters θ, (ii) the external input $\mathbf{I}$, or (iii) other auxiliary variables for the neural ODE, such that if $\Psi(\boldsymbol{d}) \geq 0$, forward invariance defined in Def. 3.1 is satisfied. Intuitively, IP casts invariance w.r.t. output specification to pose constraints on non-output in ODEs.

3.1. Invariance Propagation to Linear Layers

We start with a simple case where invariance is propagated to linear layers of the neural ODE, which normally occurs at the output layer without nonlinear activation functions.

Neural ODE Reformulation. Without loss of generality, we follow (3) and assume a linear output layer $f_{\theta_{K-1,K}}$,

$$\dot{\boldsymbol{x}} = \sum_{i=1}^{n_2} \theta_{K-1,K}^i \boldsymbol{z}_{K-1}^i = \theta_{K-1,K}^{\mathcal{P}} \boldsymbol{z}_{K-1}^{\mathcal{P}} + \theta_{K-1,K}^{\mathcal{N}} \boldsymbol{z}_{K-1}^{\mathcal{N}} \tag{8}$$

where $\boldsymbol{z}_{K-1} = (f_{\theta_{K-1,K-2}} \circ \dots \circ f_{\theta_{1,2}})(\boldsymbol{x})$, $\theta_{K-1,K}^i$ is the i'th column of $\theta_{K-1,K} \in \mathbb{R}^{n \times n_{K-1}}$, $\boldsymbol{z}_{K-1}^i$ is the i'th entry of $\boldsymbol{z}_{K-1} \in \mathbb{R}^{n_{K-1}}$, $\mathcal{P}$ and $\mathcal{N}$ describe sets of columns that are updatable parameters (that the invariance is propagated to) and constants, respectively. We drop the bias term for cleaner notation.

Propagation to Linear Layers. Our goal is to propagate the invariance to a subset of parameters. We treat $\theta_{K-1,K}^{\mathcal{P}}$ as a variable while taking other parameters $\theta_{K-1,K}^{\mathcal{N}}$ as constants. Given an arbitrary output specification $h(\boldsymbol{x}) \geq 0$, we can define a ψ_1 function in the form:

$$\psi_1(\boldsymbol{x}, \theta_{K-1,K}^{\mathcal{P}}) := \frac{dh(\boldsymbol{x})}{d\boldsymbol{x}} f_\theta(\boldsymbol{x}) + \alpha_1(h(\boldsymbol{x})), \tag{9}$$

where $\alpha_1(\cdot)$ is a class $\mathcal{K}$ function. Note that $\theta_{K-1,K}^{\mathcal{P}}$ is implicitly defined in f_θ. Combining (9) with (8), the following theorem shows the invariance of the neural ODE (1):

Theorem 3.3. *Given a neural ODE as in (8) and an output specification $h(\boldsymbol{x}) \geq 0$, if there exist a class $\mathcal{K}$ function α_1 and $\theta_{K-1,K}^{\mathcal{P}}$ such that with ψ_1 as in (9),*

$$\Psi(\theta_{K-1,K}^{\mathcal{P}} | \boldsymbol{x}) = \psi_1(\boldsymbol{x}, \theta_{K-1,K}^{\mathcal{P}}) \geq 0, \tag{10}$$

for all $\boldsymbol{x}$ such that $h(\boldsymbol{x}) \geq 0$, where $\Psi(\theta_{K-1,K}^{\mathcal{P}} | \boldsymbol{x}) = \frac{dh}{d\boldsymbol{x}} \theta_{K-1,K}^{\mathcal{P}} \boldsymbol{z}_{K-1}^{\mathcal{P}} + \frac{dh}{d\boldsymbol{x}} \theta_{K-1,K}^{\mathcal{N}} \boldsymbol{z}_{K-1}^{\mathcal{N}} + \alpha_1(h(\boldsymbol{x}))$, then the neural ODE is forward invariant.

The proof and existence of α_1 are shown in Appendix A.1.

Brief Summary. Thm. 3.3 provides a condition on the parameter $\theta_{K-1,K}^{\mathcal{P}}$ that implies the invariance of the neural ODE. In other words, by modifying the parameter $\theta_{K-1,K}^{\mathcal{P}}$ such that (10) is always satisfied, we can guarantee the invariance. The algorithm is shown in the next section. Moreover, since we only need to take the derivative of $h(\boldsymbol{x})$

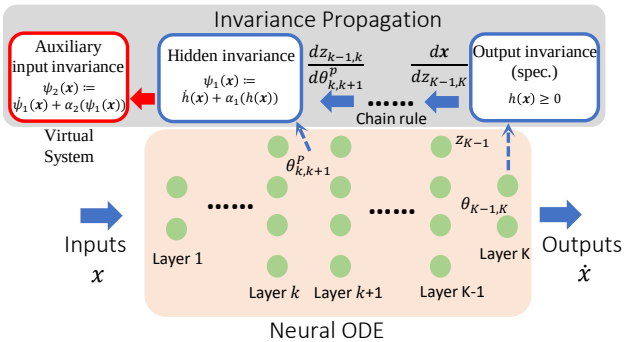

Figure 2. Invariance propagation to an arbitrary layer of the neural ODE with auxiliary virtual linear space.

once, as shown in (10), this is analogous to a first-order HOCBF (i.e., $m = 1$ in Def. 2.5).

IP to the Proper Parameters. By using the Theorem 3.3 , we can propagate the invariance to neural ODE parameters in linear layers. However, note that we need to choose the parameters such that all the output of the neural ODE are able to be changed by modifying the target parameters. Otherwise, the output specification may fail to be guaranteed.

3.2. Invariance Propagation to Nonlinear Layers

In this section, we consider how we may efficiently propagate the invariance to the weight parameters of an arbitrary layer of the neural ODE (including the output layer with nonlinear activation functions). Theoretically, we can propagate the invariance to arbitrary layers using the existing HOCBF theory. However, the resulting invariance enforcement would be nonlinear programs (i.e., the HOCBF constraint (7) will be nonlinear in $\boldsymbol{u}$), which are computationally hard and inefficient to solve. Moreover, the formulation above does not allow us to incorporate the IP in the training loop to address the conservativeness of the invariance as discussed next. Our method works for both (1) and (2), so we only consider (1) for simplicity.

An Auxiliary Linear System. Given a neural ODE (1), we want to propagate the invariance to the partial parameter (similar to (8)) at the k'th layer $\theta^{\mathcal{P}}_{k,k+1} \in \mathbb{R}^{n^{\mathcal{P}}_{k+1} \times n_k}$ with $k \in \{1, \dots, K\}$ and $n^{\mathcal{P}}_{k+1} \leq n_{k+1}$. Then, we flatten the matrix parameter $\theta^{\mathcal{P}}_{k,k+1}$ to a vector form $\theta^{\mathcal{P}}_k \in \mathbb{R}^{d^{\mathcal{P}}_k}$ row-wise, where $d^{\mathcal{P}}_k = n^{\mathcal{P}}_{k+1} n_k$ is the dimension of the vector. Instead of directly propagating the invariance to the parameter $\theta^{\mathcal{P}}_k$ and resulting nonlinear constraints, we propagate the invariance to an auxiliary linear system:

$$\dot{\theta}^{\mathcal{P}}_k = A^{\mathcal{P}}_k \theta^{\mathcal{P}}_k + B^{\mathcal{P}}_k u^{\mathcal{P}}_k \tag{11}$$

where $A^{\mathcal{P}}_k \in \mathbb{R}^{d^{\mathcal{P}}_k \times d^{\mathcal{P}}_k}, B^{\mathcal{P}}_k \in \mathbb{R}^{d^{\mathcal{P}}_k \times d^{\mathcal{P}}_k}$ are chosen such that the auxiliary system is controllable and $u^{\mathcal{P}}_k \in \mathbb{R}^{d^{\mathcal{P}}_k}$ is the auxiliary control input. The exact choice of $A^{\mathcal{P}}_k$ and $B^{\mathcal{P}}_k$ may slightly impact the performance, which is further discussed in Appendix B. This specific formulation allows performing IP on $u^{\mathcal{P}}_k$ linearly (which will become clearer later on) as opposed to directly on $\theta^{\mathcal{P}}_k$, which is susceptible to nonlinearity. An overview is illustrated in Fig. 2.

Propagation to Auxiliary System. We first propagate the invariance to parameter $\theta^{\mathcal{P}}_k$ by defining a function ψ_1 similar to (9), which is illustrated by the blue boxes in Fig. 2. Then, we further propagate the invariance to the u^p_k in system (11) by defining another function ψ_2 (the red box in Fig. 2):

$$\begin{aligned}
\psi_1(\boldsymbol{x}, \theta^{\mathcal{P}}_k) &:= \frac{dh(\boldsymbol{x})}{d\boldsymbol{x}} f_\theta(\boldsymbol{x}) + \alpha_1(h(\boldsymbol{x})), \\
\psi_2(\boldsymbol{x}, u^{\mathcal{P}}_k) &:= \frac{\partial \psi_1}{\partial \boldsymbol{x}} f_\theta(\boldsymbol{x}) + \frac{\partial \psi_1}{\partial \theta^{\mathcal{P}}_k} \dot{\theta}^{\mathcal{P}}_k + \alpha_2(\psi_1(\boldsymbol{x}, \theta^{\mathcal{P}}_k)),
\end{aligned} \tag{12}$$

where $\alpha_1(\cdot), \alpha_2(\cdot)$ are class $\mathcal{K}$ functions and ψ_1, ψ_2 are defined in the similar spirit to (5). Remark that different from (9), here, ψ_1 is nonlinear regarding $\theta^{\mathcal{P}}_k$ yet the newly introduced ψ_2 is linear to the auxiliary variable $u^{\mathcal{P}}_k$ thanks to $\dot{\theta}^{\mathcal{P}}_k$ depicting a linear system w.r.t. $u^{\mathcal{P}}_k$ as shown in (11). Combining (12) with (11), the following theorem shows the invariance:

Theorem 3.4. *Given a neural ODE defined by (1) and an output specification $h(\boldsymbol{x}) \geq 0$, if there exist class $\mathcal{K}$ functions α_1, α_2 and $u^{\mathcal{P}}_k$ such that with ψ_1, ψ_2 as in (12),*

$$\Psi(u^{\mathcal{P}}_k | \boldsymbol{x}) = \psi_2(\boldsymbol{x}, u^{\mathcal{P}}_k) \geq 0, \tag{13}$$

for all $\boldsymbol{x}$ that satisfies $h(\boldsymbol{x}) \geq 0$ and $\psi_1(\boldsymbol{x}) \geq 0$, where $\Psi(u^{\mathcal{P}}_k | \boldsymbol{x}) = \frac{d^2 h(\boldsymbol{x})}{d\boldsymbol{x}^2} f^2_\theta(\boldsymbol{x}) + \frac{dh(\boldsymbol{x})}{d\boldsymbol{x}} \frac{\partial f_\theta(\boldsymbol{x})}{\partial \theta^{\mathcal{P}}_k} (A^{\mathcal{P}}_k \theta^{\mathcal{P}}_k + B^{\mathcal{P}}_k u^{\mathcal{P}}_k) + (\frac{dh(\boldsymbol{x})}{d\boldsymbol{x}} \frac{\partial f_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}} + \frac{d\alpha_1(h(\boldsymbol{x}))}{d\boldsymbol{x}}) f_\theta(\boldsymbol{x}) + \alpha_2(\psi_1(\boldsymbol{x}))$, then the neural ODE is forward invariant.

The proof and existence of α_1, α_2 are shown in Appendix A.2. Intuitively, invariance is first propagated via ψ_1 to $\theta^{\mathcal{P}}_k$, then via ψ_2 to $u^{\mathcal{P}}_k$, rendering a linear constraint in (13). We will further show how this enforces invariance in Sec. 4.2.

Brief Summary. Note that (13) is linear in $u^{\mathcal{P}}_k$ with the assistance of system (11). Instead of directly changing the parameters of the neural ODE for the invariance as Sec. 3.1, we find auxiliary control $u^{\mathcal{P}}_k$ that satisfies the constraint (13) to dynamically change the parameters. Also, since we take the derivative of $h(\boldsymbol{x})$ twice, as in (13), this is analogous to a second-order HOCBF (i.e., $m = 2$ in Def. 2.5)

IP to the Proper Parameters. While Thm. 3.4 allows us to propagate the invariance to arbitrary neural ODE parameters, the choice of the parameters may affect the performance, e.g., the model's accuracy). The specific parameter choice depends on the model structure and the task's output specification. In most cases, we may wish to choose the parameters

of the same layer to propagate the invariance to. However, it is also possible to choose parameters of different layers, as long as we define auxiliary dynamics for all the parameters as in (11). The proposed method still works in such cases. We may need to choose the parameters such that the output of the neural ODE can all be changed, as in the linear case.

3.3. Invariance Propagation to External Input

We consider a neural ODE in the form of (2) with an external input $\mathbf{I}$.

Approach 1: As in Sec. 3.1, we may directly reformulate (2) in the following affine form:

$$\dot{\boldsymbol{x}} = f_\theta(\boldsymbol{x}) + g_\theta(\boldsymbol{x})\mathbf{I}, \tag{14}$$

where f_θ is defined as in (1), $g_\theta : \mathbb{R}^n \to \mathbb{R}^{n\times n_\mathbf{I}}$ is another neural network parameterized by θ. Then, we can use the similar technique as in Sec. 3.1 to propagate the invariance to the external input $\mathbf{I}$ as they are both in affine forms.

Approach 2: If we do wish to keep neural ODEs with external input as in the form of (2), then we may define auxiliary linear dynamics as in Sec. 3.2, and augment (2) by the following form:

$$\dot{\boldsymbol{x}} = f_\theta^{'}(\boldsymbol{x}, \boldsymbol{y}), \qquad \dot{\boldsymbol{y}} = A\boldsymbol{y} + B\mathbf{I}, \tag{15}$$

where $\boldsymbol{y} \in \mathbb{R}^{n_\mathbf{I}}$ is the auxiliary variable, $A \in \mathbb{R}^{n_\mathbf{I}\times n_\mathbf{I}}, B \in \mathbb{R}^{n_\mathbf{I}\times n_\mathbf{I}}$ are defined such that the linear system is controllable (similar to (11)). Then, we can use the similar technique as in Sec. 3.2 to propagate the invariance to the external input $\mathbf{I}$ via the auxiliary variable $\boldsymbol{y}$. In fact, the above neural ODE becomes a stacked neural ODE, which will be further studied (as discussed in Appendix E).

4. Enforcing Invariance in Neural ODEs

Here, we show how we proceed from the theoretical framework in Sec. 3 to efficient algorithms of IP on neural ODEs.

4.1. Algorithms for Linear layers

Enforcing invariance. Enforcing the invariance of a neural ODE is equivalent to the satisfaction of the condition in Thm. 3.3. Also by proof in Appendix A.1, we can always find a class $\mathcal{K}$ function $\alpha_1(\cdot)$ such that there exists $\theta^{\mathcal{P}}_{K-1,K}$ that makes (10) satisfied if $h(\boldsymbol{x}(t_0)) \geq 0$. If $h(x_1(t_0)) \leq 0$, then the output of the neural ODE will be driven to satisfy $h(\boldsymbol{x}) \geq 0$ when the constraint (10) in Thm. 3.3 is satisfied due to its Lyapunov property (Ames et al., 2012). The enforcing of the invariance could vary in different applications and we do not restrict to exact methods. We provide a minimum-deviation quadratic program (QP) approach.

Let $\theta^{\mathcal{P}\dagger}_{K-1,K} \in \mathbb{R}^{n\times n^{\mathcal{P}}_{K-1}}$ denote the value of $\theta^{\mathcal{P}}_{K-1,K}$ during training or after training. Then, we can formulate the following optimization:

$$\theta^{\mathcal{P}*}_{K-1,K} = \arg\min_{\theta^{\mathcal{P}}_{K-1,K}} ||\theta^{\mathcal{P}}_{K,K-1} - \theta^{\mathcal{P}\dagger}_{K-1,K}||^2, \text{ s.t. (10)}, \tag{16}$$

where $||\cdot||$ denotes the Euclidean norm. The above optimization becomes a QP with all other variables fixed except $\theta^{\mathcal{P}}_{K-1,K}$. This solving method has been shown to work in (Ames et al., 2017) (Glotfelter et al., 2017) (Xiao & Belta, 2022). At each discretization step, we solve the above QP and get $\theta^{\mathcal{P}*}_{K-1,K}$. Then we set $\theta^{\mathcal{P}}_{K-1,K} = \theta^{\mathcal{P}*}_{K-1,K}$ during the inference of the neural ODE. This way, we can enforce the invariance, i.e., guarantee that $h(\boldsymbol{x}(t)) \geq 0, \forall t \geq t_0$. The process is summarized in Algorithm 1.

Complexity of Enforcing Invariance. The computational complexity of the QP (16) is $\mathcal{O}(q^3)$, where $q = n^{\mathcal{P}}_{K-1}n$. When there is a set S of output specifications, we just add the corresponding constraint (10) for each specification to (16), and the number of constraints will not significantly increase the complexity. It is also possible to get the closed-form solution of the QP (Ames et al., 2017) when there are only a few output specifications.

4.2. Algorithms for Nonlinear layers

Stability of Auxiliary Systems. In this case, we need to make sure that the parameter $\theta^{\mathcal{P}}_k$ of the neural ODE is stabilized as it is dynamically controlled by (11). To enforce this, we use control Lyapunov functions (CLFs) (Ames et al., 2012). Specifically, for each $\theta^{\mathcal{P}}_{k_j}, j \in \{1\ldots, d^{\mathcal{P}}_k\}$, where $\theta^{\mathcal{P}}_{k_j}$ is a component of $\theta^{\mathcal{P}}_k$, we define a CLF $V(\theta^{\mathcal{P}}_{k_j}) = (\theta^{\mathcal{P}}_{k_j} - \theta^{\mathcal{P}\dagger}_{k_j})^2$, where $\theta^{\mathcal{P}\dagger}_{k_j}$ is the value of $\theta^{\mathcal{P}}_{k_j}$ during or after training. Then, any $u^{\mathcal{P}}_k$ that satisfies:

$$\Phi(u^{\mathcal{P}}_k|\theta^{\mathcal{P}}_{k_j}) \leq 0, j \in \{1, \ldots, d^{\mathcal{P}}_k\}, \tag{17}$$

where $\Phi(u^{\mathcal{P}}_k|\theta^{\mathcal{P}}_{k_j}) = \frac{dV(\theta^{\mathcal{P}}_{k_j})}{d\theta^{\mathcal{P}}_{k_j}}(A^{\mathcal{P}}_{k_j}\theta^{\mathcal{P}}_k + B^{\mathcal{P}}_{k_j}u^{\mathcal{P}}_k) + \epsilon_j V(\theta^{\mathcal{P}}_{k_j})$, $A^{\mathcal{P}}_{k_j} \in \mathbb{R}^{1\times d^{\mathcal{P}}_k}, B^{\mathcal{P}}_{k_j} \in \mathbb{R}^{1\times d^{\mathcal{P}}_k}$ are the j'th rows of $A^{\mathcal{P}}_k, B^{\mathcal{P}}_k$ in (11), respectively and $\epsilon_j > 0$, will render the auxiliary systems (11) stable. The proof is in Appendix A.3.

Enforcing invariance. Enforcing the invariance of a neural ODE is equivalent to the satisfaction of the condition in Thm. 3.4. By proof in Appendix A.2, we can always find class $\mathcal{K}$ functions α_1, α_2 such that there exists $u^{\mathcal{P}}_k$ that makes (13) satisfied if $h(\boldsymbol{x}(t_0)) > 0$. Again, we provide a minimum-deviation quadratic program (QP) approach:

$$(u^{\mathcal{P}*}_k, \delta^*_{1:d^{\mathcal{P}}_k}) = \arg\min_{u^{\mathcal{P}}_k, \delta_{1:d^{\mathcal{P}}_k}} ||u^{\mathcal{P}}_k||^2 + \sum_{j=1}^{d^{\mathcal{P}}_k} w_j\delta_j^2,$$
$$\text{s.t. (13) and } \Phi(u^{\mathcal{P}}_k|\theta^{\mathcal{P}}_{k_j}) \leq \delta_j, j \in \{1, \ldots, d^{\mathcal{P}}_k\}, \tag{18}$$

Algorithm 1 Invariance Propagation to Parameters

Input: Output specification set S, trained or in-training neural ODE (1).
(a) Choose the model parameters $\theta_k^{\mathcal{P}}$ that we wish to propagate the invariances (from the set S) to.
(b) Make the chosen parameters $\theta_k^{\mathcal{P}}$ as symbolic variables.
(c) Make $f_\theta(\boldsymbol{x})$ in the neural ODE (1) as a symbolic function in terms of $\theta_k^{\mathcal{P}}$ and $\boldsymbol{x}$.
if *Propagate to nonlinear layers* **then**
  (d) Define controllable linear dynamics (11) for $\theta_k^{\mathcal{P}}$.
  (e) Propagate invariances to $u_k^{\mathcal{P}}$ of system (11) by (13).
  (f) Define CLFs to stabilize $\theta_k^{\mathcal{P}}$ by (17).
  (g) Formulate the QP (18).
else
  (d) Formulate the QP (16).
end if
repeat
  Get the trained or in-training value $\theta_k^{\mathcal{P}\dagger}$ of $\theta_k^{\mathcal{P}}$.
  if *Propagate to nonlinear layers* **then**
    Solve the QP (18) and get $u_k^{\mathcal{P}*}$.
    Get $\theta_k^{\mathcal{P}*}$ by integrating (11) with $u_k^{\mathcal{P}} = u_k^{\mathcal{P}*}$ ($u_k^{\mathcal{P}}$ is piecewise constant).
  else
    Get $\theta_k^{\mathcal{P}*}$ by solving the QP (16).
  end if
  Set $\theta_k^{\mathcal{P}} = \theta_k^{\mathcal{P}*}$ for the neural ODE (1).
until $Training$ or $Inference$ is $done$

where $\delta_j \in \mathbb{R}$ is a slack variable that makes the CLF constraint soft (not conflict with (13)), and $w_j > 0, j \in \{1, \ldots, d_k^{\mathcal{P}}\}$ are pre-defined coefficients of penalties on the relaxations. The above optimization becomes a QP with all other variables fixed except $u_k^{\mathcal{P}}, \delta_j$, as discussed at the end of the last subsection. At each discretization step, we solve the above QP and get $u_k^{\mathcal{P}*}$. Then, the optimal parameter $\theta_k^{\mathcal{P}^*}$ is determined by the integration of (11) with $u_k^{\mathcal{P}} = u_k^{\mathcal{P}*}$, and set $\theta_k^{\mathcal{P}} = \theta_k^{\mathcal{P}^*}$ during the inference. This way, we can enforce the invariance, i.e., guarantee that $h(\boldsymbol{x}(t)) \geq 0, \forall t \geq t_0$. We summarize the process in Algorithm 1.

Complexity of enforcing invariance The computational complexity of the QP (18) is $\mathcal{O}(q^3)$, where $q = 2d_k^{\mathcal{P}}$. When there is a set S of output specifications, we just add the corresponding constraint (13) for each specification to (18). The complexity is a little higher than the one in the case of invariance enforcement in a linear layer. This is due to the fact that the dimension of decision variables is doubled. Nonetheless, this is still efficient to solve.

4.3. Training Neural ODEs with Invariance.

The invariance of neural ODEs can be enforced even after the training of neural ODEs. However, we need to hand-tune the parameters of class $\mathcal{K}$ functions in the QP (16) or (18) to address the conservativeness of this approach, which is non-trivial when we have many output specifications.

We leverage the power of differentiable QP (Amos & Kolter, 2017). *For neural ODEs whose invariance are enforced on the external input* **I**, the differentiable QP that enforces the invariance is stacked to the neural ODE, and thus, the training is performed via the standard stochastic gradient descent. *For neural ODEs whose invariance are enforced on the model parameter* θ (θ is either $\theta_{K-1,K}^{\mathcal{P}}$ as in Sec. 4.1 or $\theta_k^{\mathcal{P}}$ as in Sec. 4.2), it is challenging to train both the neural ODE and differentiable QP simultaneously in the same pipeline. Thus, we propose the following two-stage training method: In the first stage, we train the neural ODE as usual and thus optimize the weight $\theta^\dagger$ of the network. In the second stage, we train via the differentiable QP (more precisely, the parameter of class $\mathcal{K}$ functions in QP (16), (18)) such that θ minimally deviates from $\theta^\dagger$. The training of the network and the QP can be performed alternatively. We summarize the training process in Fig. 5 in Appendix C.

5. Experiments

We set up experiments to answer the following questions:

- Does our algorithm match the theoretical potential in various learning tasks both quantitatively and qualitatively?
- How does our invariance propagation compare with state-of-the-art approaches for enforcing output specifications?
- How does our proposed method scale with the number of parameters of the neural ODE and handle complex specifications of dynamical systems?

5.1. Spiral Curve Regression with Specifications

In this experiment, we aim to impose trajectory constraints on the spiral dynamics task proposed in (Chen et al., 2018). The training data comes from solving an ODE $[\dot{x}, \dot{y}]^T = A[x^3, y^3]^T$, where $A = [-0.1, -2.0; 2.0, -0.1]$. We use a neural ODE to fit the data. We additionally require the trajectory $\boldsymbol{x} = (x, y)$ to avoid some areas defined by $h_j(\boldsymbol{x}) \geq 0, j \in S$, where S denotes a set of constraints.

Comparison between invariance enforcing on different layers. We first compare the performance of neural ODEs when propagating invariances to different layers and different numbers of model parameters. As shown in Table 1, The output specifications are all guaranteed (Sat. ≥ 0) with invariance propagation, which shows the flexibility of the method. While the output specifications are violated in pure neural ODEs. The computation time is higher when we propagate the invariances to the hidden layer than one of the output layers, and the computation time slightly increases when we significantly increase the number of chosen parameters, although the inference errors do not actually vary

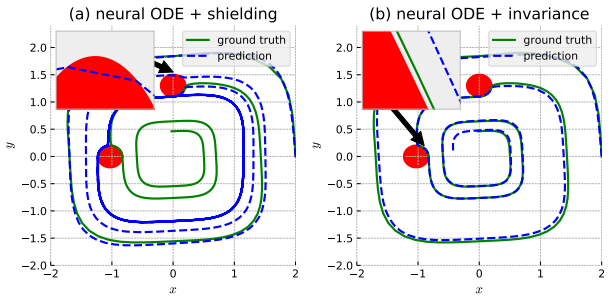

Figure 3. Invariance in training for spiral curve regression with output specifications. The shielding method can only guarantee point-wise satisfaction, and thus the specification can still be violated between samplings.

too much. An illustrative example is shown in Figs. 7-8 in Appendix F.1.

Table 1. Spiral curve comparisons between neural ODE and invariance propagated to different layers. The numbers in brackets denote the number of parameters chosen in the invariance. Sat. (Satisfaction) ≥ 0 denotes the satisfaction of output specifications.

METHOD	SAT. (≥ 0)	MSE (↓)	TIME (↓)
NEURAL ODE	-0.031	0.510	0.003
HIDDEN INV. (6)	0.003	0.393	0.025
HIDDEN INV. (20)	0.001	0.391	0.030
HIDDEN INV. (60)	$4e^{-4}$	0.387	0.044
HIDDEN INV. (100)	$9e^{-4}$	0.391	0.057
OUTPUT INV. (6)	$2e^{-4}$	0.441	0.007
OUTPUT INV. (20)	$2e^{-4}$	0.442	0.010
OUTPUT INV. (60)	$2e^{-4}$	0.442	0.017
OUTPUT INV. (100)	$2e^{-4}$	0.442	0.025

Comparison with benchmarks. The proposed invariance allows us to enforce arbitrary specifications after training with a different number of parameters. An illustrative example is shown in Fig. 9 in Appendix F.1. With invariance in the training loop, the model outputs can strictly satisfy the output specifications while staying close to the ground truth (see Figure 3b). The comparisons between our proposed (output) invariance with other benchmarks are shown in Table 2 in which the results are evaluated using 100 trained models for each method. Compared to the shielding method (Ferlez et al., 2020), our invariance model can achieve better performance. Most importantly, the proposed invariance can guarantee more complex specifications including addressing the inter-sampling effect, i.e., specification satisfaction between sampling time, as shown in Fig. 3. Both the filter approach (Pereira et al., 2020) and BarrierNet (Xiao et al., 2023) perform badly when they are placed outside the neural ODE (less computationally expensive than the case when they are placed inside the neural ODE). Moreover, the filter approach is not trainable, which may introduce the worse performance. In contrast, the performance of our proposed invariance is similar when the QP is placed inside and outside the neural ODE (we only show the result when the QP is placed inside the neural ODE in Table 2), which shows its flexibility.

Table 2. Spiral curve comparisons with benchmarks. [I]/[O] denotes the Filter (Pereira et al., 2020) or BarrierNet (BNet) (Xiao et al., 2023) is inside/outside the neural ODE (Chen et al., 2018). Items are short for In-loop training test mean-squared error (IN-LOOP MSE), Post-training test mean-squared error (POST MSE), Complex specifications and Inter-sampling effect (COMP. & IS), Pointwise guarantee (PW GUAR.), respectively. Shielding is (Ferlez et al., 2020). The method items are short for Neural ODE (NEUR. ODE), Neural ODE with low conservative training (NODE-L), Neural ODE with high conservative training (NODE-H), respectively.

METHOD	IN-LOOP MSE(↓)	POST MSE(↓)	COMP. & IS	PW GUAR.
NEUR. ODE	0.49±0.17	0.49±0.17	×	×
NODE-L	0.59±0.18	0.59±0.18	×	×
NODE-H	0.73±0.18	0.73±0.18	×	×
SHIELDING	0.61±0.12	0.61±0.10	×	√
FILTER [I]	N/A	0.55±0.21	√	√
FILTER [O]	N/A	1.27±0.14	√	√
BNET [I]	0.64±0.09	0.44±0.08	√	√
BNET [O]	N/A	1.04±0.11	√	√
INV. (OURS)	0.55±0.12	0.44±0.08	√	√

5.2. Convexity Portrait of a Function

In this experiment, we assess whether our method can enforce that the neural ODE outputs satisfy Jensen's inequality. Jensen's inequality can be used to characterize whether a function is convex or not. In other words, a function g is convex if the Jensen's inequality is satisfied: $\mu_1 g(x) + \mu_2 g(y) \geq g(\mu_1 x + \mu_2 y)$, where $\mu_1 \in [0, 1], \mu_2 \in [0, 1]$ such that $\mu_1 + \mu_2 = 1$. A neural ODE is not guaranteed to satisfy Jensen's inequality as illustrated by the red-dashed curve in Figure 10b (in appendix). However, with the proposed (hidden and output) invariances, the model outputs are guaranteed to satisfy Jensen's inequality, as shown by the blue-dashed and cyan dashed curves in Figure 10b (in appendix).

5.3. HalfCheetah-v2 and Walker2d-v2

In this section, we evaluate our invariance framework on two publicly available datasets for modeling physical dynamical systems (Lechner & Hasani, 2022; Hasani et al., 2021). The two datasets consist of trajectories of the HalfCheetah-v2 and Walker2d-v2 3D robot systems (Brockman et al., 2016) generated by the Mujoco physics engine (Todorov et al., 2012). Each trajectory represents a sequence of a

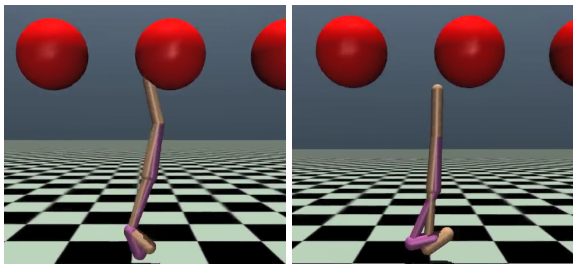

Figure 4. Planning for Walker2d-v2 in obstacle avoidance using neural ODE (left) and invariance (right). Collisions happen when using neural ODE only, while they can be avoided with guarantees using the proposed invariance method.

17-dimensional vector describing the system's state, such as the robot's joint angles and poses. For each of the two tasks, we define 34 safety constraints that restrict the system's evolution to the value ranges observed in the dataset. We compare our invariance approach with a hard truncation of the system state, i.e., projecting points violating the constraints to the nearest points that satisfy them. Our invariance framework can achieve competitive performance compared to other approaches, as shown in Table 3, while guaranteeing the satisfaction of complex safety specifications. We enforced our invariance on 17, 34, and 170 model parameters, respectively. We also present a case study (Fig. 4) in planning for obstacle avoidance for the Walker2d-v2 in which the truncation method fails to work.

Table 3. Walker2d-v2 and halfcheetah-v2 comparisons between Neural ODE (Chen et al., 2018), Truncation (Brockman et al., 2016), and Invariance (ours). Items are short for Walker2d-v2 test mean-squared error (W2D MSE), halfcheetah-v2 test mean-squared error (HC MSE), Complex specifications and Inter-sampling effect (COMP. & IS), Joint limit satisfaction (SAFETY), respectively.

METHOD	W2D MSE (↓)	HC MSE (↓)	COMP. & IS	SAFETY (≥ 0)
NEUR. ODE	1.06±0.07	2.17±0.03	×	-1.78
TRUNCATION	1.15±0.08	2.17±0.03	×	-8.13
INV. (OURS)	1.06±0.07	2.13±0.02	√	0.0

5.4. Lidar-based End-to-End Autonomous Driving

In this section, we consider Lidar-based end-to-end autonomous driving that has complex specifications from dynamics. The approach of finding neural ODE specifications from dynamics can be found in Appendix D. The neural ODE takes a Lidar point cloud as input **I**, and outputs controls for the autonomous vehicle to follow the lane. The problem and training setup is shown in Appendix F.4.

With noisy Lidar, the neural ODE controller may cause the ego vehicle to collide with the other moving vehicle during the overtaking process (the red trajectory shown in Figure 13 in Appendix F.4). Although with safety guarantees, the resulting trajectory from a safe filter (Pereira et al., 2020) may make the ego vehicle conservative (as the blue trajectory shown in Figure 13), and thus stay unnecessarily far away from the optimal trajectory (ground truth). The BarrierNet (Xiao et al., 2023) is also a filter, but it addresses conservativeness by including the CBF (filter) in the training loop. However, the training of a BarrierNet is harder compared with the invariance as reference controls and the relative weight among them should also be trained in addition to the CBF parameters. We summarize this comparison in Table 4 that includes testing results of 100 case studies under noisy lidar perception. The invariance has the least conservativeness while guaranteeing safety.

Table 4. Self-driving comparisons between safe filter (Pereira et al., 2020), BarrierNet (Xiao et al., 2023), neural ODE (Chen et al., 2018) and invariance (ours). Items are short for Trajectory test mean-squared error (TRAJ. MSE), Conservativeness measurement (CONSER.), Safety measurement (SAFETY), Model complexity (MOD. CMP.), respectively. L, H are shorts for Low and High.

METHOD	TRAJ. MSE (↓)	CONSER. (≥ 0 & ↓)	SAFETY (≥ 0)	DY. FREE	MOD. CMP.
NEUR. ODE	0.46±0.04	−13.1±1.49	−17.26	√	L
SAFE FILTER	0.96±0.04	27.69±1.08	24.60	×	H
BARRIERNET	0.34±0.01	8.51±0.33	7.69	×	H
INV. (OURS)	0.36±0.01	1.97±0.06	1.83	√	L

6. Related Works

Neural ODEs for imitation learning. Neural ODEs (Chen et al., 2018) (Chen et al., 2020) are powerful dynamical systems modeling tools, widely used in applications to learning system kinetics (Kim et al., 2021) (Alvarez et al., 2020) (Baker et al., 2022), in graphics (Asikis et al., 2022), in discovering novel materials (Chen et al., 2022), and in robot controls (Hasani et al., 2017; Amini et al., 2020; Lechner & Hasani, 2022; Lechner et al., 2020; Vorbach et al., 2021). Neural ODEs are continuous-time universal approximators (Kidger et al., 2020) that perform competitive to their static and discretized neural network counterparts, once their complexity issues (Massaroli et al., 2020) are resolved by better numerical solvers (Poli et al., 2020), or by their closed-form variants (Hasani et al., 2022). Recent methods provide safety guarantees for inference in a neural ODE system, e.g. stochastic reachability analysis (Gruenbacher et al., 2020). However, there are no methods to simultaneously train the model while guaranteeing safety. Here, we address this issue by forward-invariance of neural ODEs.

Set invariance and CBFs. An invariant set has been widely used to characterize the safe behavior of dynamical systems (Preindl, 2016) (Rakovic et al., 2005) (Ames et al., 2017)

(Glotfelter et al., 2017) (Xiao et al., 2023). In the state of the art, Control Barrier Functions (CBFs) are also widely used to prove set invariance (Aubin, 2009), (Prajna et al., 2007), (Wisniewski & Sloth, 2013). They can be traced back to optimization problems (Boyd & Vandenberghe, 2004), and are Lyapunov-like functions (Tee et al., 2009), (Wieland & Allgöwer, 2007). Existing CBF approaches have significant limitations: They fail on systems with unknown dynamics, provide rather conservative guarantees, and work efficiently only for affine systems. Our work addresses all these limitations.

Existing approaches for guarantees in neural networks. Recent advances in differentiable optimization methods show promise for safety-guaranteed neural network controllers (Pereira et al., 2020; Amos et al., 2018; Xiao et al., 2023; Wang et al., 2023). The differentiable optimizations are usually served as a layer (filter) in the neural networks. In (Amos & Kolter, 2017), a differentiable quadratic program (QP) layer, called OptNet, was introduced. OptNet with CBFs has been used in neural networks as a filter for safe controls (Pereira et al., 2020), in which CBFs are not trainable, thus, potentially limiting the system's learning performance. In (Deshmukh et al., 2019; Jin et al., 2020; Zhao et al., 2021; Ferlez et al., 2020), safety guaranteed neural network controllers have been learned through verification-in-the-loop training. The verification approaches cannot ensure coverage of the entire state space. While the proposed invariance can avoid such issues, and generalize to a wide class of guarantees.

7. Conclusions, Discussions and Future Work

We have demonstrated the effectiveness of our invariance propagation method in ensuring the safe operation of neural ODE instances in a series of dynamical system learning tasks. Nonetheless, our method faces a few shortcomings which provide directions for future work to focus on.

In particular, our method requires neural ODEs to have continuously differentiable activation functions. Moreover, our method requires prior knowledge of output specifications. Future work may investigate how to learn specifications from observational data for unknown specifications. Finally, we observed the propagation of invariance is subjected to the vanishing gradient for deeper networks. We can alleviate this shortcoming by gradient preservation methods such as mixed-memory ODE-based networks (Lechner & Hasani, 2022) which we will investigate in future work.

Acknowledgements

The research was supported in part by Capgemini Engineering. It was also partially sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. This research was also supported in part by the AI2050 program at Schmidt Futures (Grant G-965 22-63172),

References

Alvarez, V. M. M., Roşca, R., and Fălcuţescu, C. G. Dynode: Neural ordinary differential equations for dynamics modeling in continuous control. *arXiv preprint arXiv:2009.04278*, 2020.

Ames, A. D., Galloway, K., and Grizzle, J. W. Control lyapunov functions and hybrid zero dynamics. In *Proc. of 51rd IEEE Conference on Decision and Control*, pp. 6837–6842, 2012.

Ames, A. D., Xu, X., Grizzle, J. W., and Tabuada, P. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2017.

Amini, A., Gilitschenski, I., Phillips, J., Moseyko, J., Banerjee, R., Karaman, S., and Rus, D. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robotics and Automation Letters*, 5(2):1143–1150, 2020.

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, pp. 136–145, 2017.

Amos, B., Rodriguez, I. D. J., Sacks, J., Boots, B., and Kolter, J. Z. Differentiable mpc for end-to-end planning and control. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 8299–8310. Curran Associates Inc., 2018.

Asikis, T., Böttcher, L., and Antulov-Fantulin, N. Neural ordinary differential equation control of dynamics on graphs. *Phys. Rev. Research*, 4:013221, Mar 2022. doi: 10.1103/PhysRevResearch.4.013221.

Aubin, J.-P. *Viability theory*. Springer, 2009.

Baker, J., Cherkaev, E., Narayan, A., and Wang, B. Learning pod of complex dynamics using heavy-ball neural odes. *arXiv preprint arXiv:2202.12373*, 2022.

Boyd, S. P. and Vandenberghe, L. *Convex optimization*. Cambridge university press, New York, 2004.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 6572–6583, 2018.

Chen, R. T., Amos, B., and Nickel, M. Learning neural event functions for ordinary differential equations. *arXiv preprint arXiv:2011.03902*, 2020.

Chen, X., Araujo, F. A., Riou, M., Torrejon, J., Ravelosona, D., Kang, W., Zhao, W., Grollier, J., and Querlioz, D. Forecasting the outcome of spintronic experiments with neural ordinary differential equations. *Nature communications*, 13(1):1–12, 2022.

Deshmukh, J. V., Kapinski, J. P., Yamaguchi, T., and Prokhorov, D. Learning deep neural network controllers for dynamical systems with safety guarantees: Invited paper. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, 2019.

Ferlez, J., Elnaggar, M., Shoukry, Y., and Fleming, C. Shieldnn: A provably safe nn filter for unsafe nn controllers. *preprint arXiv:2006.09564*, 2020.

Glotfelter, P., Cortes, J., and Egerstedt, M. Nonsmooth barrier functions with applications to multi-robot systems. *IEEE control systems letters*, 1(2):310–315, 2017.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Gruenbacher, S., Hasani, R., Lechner, M., Cyranka, J., Smolka, S. A., and Grosu, R. On the verification of neural odes with stochastic guarantees. *arXiv preprint arXiv:2012.08863*, 2020.

Gu, A., Goel, K., and Re, C. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2021.

Haber, E. and Ruthotto, L. Stable architectures for deep neural networks. *Inverse problems*, 34(1):014004, 2017.

Hasani, R., Lechner, M., Amini, A., Rus, D., and Grosu, R. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7657–7666, 2021.

Hasani, R., Lechner, M., Amini, A., Liebenwein, L., Ray, A., Tschaikowski, M., Teschl, G., and Rus, D. Closed-form continuous-time neural networks. *Nature Machine Intelligence*, pp. 1–12, 2022.

Hasani, R. M., Haerle, D., Baumgartner, C. F., Lomuscio, A. R., and Grosu, R. Compositional neural-network modeling of complex analog circuits. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2235–2242. IEEE, 2017.

Jin, W., Wang, Z., Yang, Z., and Mou, S. Neural certificates for safe control policies. *preprint arXiv:2006.08465*, 2020.

Khalil, H. K. *Nonlinear Systems*. Prentice Hall, third edition, 2002.

Kidger, P., Morrill, J., Foster, J., and Lyons, T. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33: 6696–6707, 2020.

Kim, S., Ji, W., Deng, S., Ma, Y., and Rackauckas, C. Stiff neural ordinary differential equations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(9):093122, 2021.

Lechner, M. and Hasani, R. Mixed-memory rnns for learning long-term dependencies in irregularly sampled time series. In *NeurIPS 2022 Memory in Artificial and Real Intelligence workshop*, 2022.

Lechner, M., Hasani, R., Amini, A., Henzinger, T. A., Rus, D., and Grosu, R. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2(10): 642–652, 2020.

Liebenwein, L., Hasani, R., Amini, A., and Rus, D. Sparse flows: Pruning continuous-depth models. *Advances in Neural Information Processing Systems*, 34, 2021.

Massaroli, S., Poli, M., Park, J., Yamashita, A., and Asama, H. Dissecting neural odes. *Advances in Neural Information Processing Systems*, 33:3952–3963, 2020.

Nagumo, M. Über die lage der integralkurven gewöhnlicher differentialgleichungen. In *Proceedings of the Physico-Mathematical Society of Japan. 3rd Series. 24:551-559*, 1942.

Pereira, M. A., Wang, Z., Exarchos, I., and Theodorou, E. A. Safe optimal control using stochastic barrier functions and deep forward-backward sdes. In *Conference on Robot Learning*, 2020.

Poli, M., Massaroli, S., Yamashita, A., Asama, H., and Park, J. Hypersolvers: Toward fast continuous-depth models.

Advances in Neural Information Processing Systems, 33: 21105–21117, 2020.

Pontryagin, L. S. *Mathematical theory of optimal processes*. Routledge, 2018.

Prajna, S., Jadbabaie, A., and Pappas, G. J. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Transactions on Automatic Control*, 52 (8):1415–1428, 2007.

Preindl, M. Robust control invariant sets and lyapunov-based mpc for ipm synchronous motor drives. *IEEE Transactions on Industrial Electronics*, 63(6):3925–3933, 2016.

Rakovic, S. V., Kerrigan, E. C., Kouramas, K. I., and Mayne, D. Q. Invariant approximations of the minimal robust positively invariant set. *IEEE Transactions on automatic control*, 50(3):406–410, 2005.

Rubanova, Y., Chen, R. T., and Duvenaud, D. K. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Tee, K. P., Ge, S. S., and Tay, E. H. Barrier lyapunov functions for the control of output-constrained nonlinear systems. *Automatica*, 45(4):918–927, 2009.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Vorbach, C., Hasani, R., Amini, A., Lechner, M., and Rus, D. Causal navigation by continuous-time neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Wang, T.-H., Xiao, W., Chahine, M., Amini, A., Hasani, R., and Rus, D. Learning stability attention in vision-based end-to-end driving policies. *arXiv preprint arXiv:2304.02733*, 2023.

Wieland, P. and Allgöwer, F. Constructive safety using control barrier functions. In *Proc. of 7th IFAC Symposium on Nonlinear Control System*, 2007.

Wisniewski, R. and Sloth, C. Converse barrier certificate theorem. In *Proc. of 52nd IEEE Conference on Decision and Control*, pp. 4713–4718, Florence, Italy, 2013.

Xiao, W. and Belta, C. Control barrier functions for systems with high relative degree. In *Proc. of 58th IEEE Conference on Decision and Control*, pp. 474–479, Nice, France, 2019.

Xiao, W. and Belta, C. High-order control barrier functions. *IEEE Transactions on Automatic Control*, 67(7):3655–3662, 2022.

Xiao, W., Wang, T.-H., Hasani, R., Chahine, M., Amini, A., Li, X., and Rus, D. Barriernet: Differentiable control barrier functions for learning of safe robot control. *IEEE Transactions on Robotics*, 2023.

Zhao, H., Zeng, X., Chen, T., Liu, Z., and Woodcock, J. Learning safe neural network controllers with barrier certificates. *Form Asp Comp*, 33:437–455, 2021.

A. Proof

A.1. Proof of Theorem 3.3 (Invariance Propagation to Linear Layers)

Given a continuously differentiable constraint $h(x) \geq 0$ ($h(\boldsymbol{x}(t_0)) \geq 0$), by Nagumo's theorem (Nagumo, 1942), the necessary and sufficient condition for the satisfaction of $h(x(t)) \geq 0, \forall t \geq t_0$ is

$$\dot{h}(x) \geq 0, \text{ when } h(x) = 0$$

The neural ODE is reformulated as

$$\dot{\boldsymbol{x}} = \sum_{i=1}^{n_2} \theta^i_{K-1,K} \boldsymbol{z}^i_{K-1} = \theta^{\mathcal{P}}_{K-1,K} \boldsymbol{z}^{\mathcal{P}}_{K-1} + \theta^{\mathcal{N}}_{K-1,K} \boldsymbol{z}^{\mathcal{N}}_{K-1}$$

and the condition in the theorem is given by

$$\Psi(\theta^{\mathcal{P}}_{K-1,K}|\boldsymbol{x}) = \underbrace{\frac{dh(\boldsymbol{x})}{d\boldsymbol{x}} \theta^{\mathcal{P}}_{K-1,K} \boldsymbol{z}^{\mathcal{P}}_{K-1} + \frac{dh(\boldsymbol{x})}{d\boldsymbol{x}} (\theta^{\mathcal{N}}_{K-1,K} \boldsymbol{z}^{\mathcal{N}}_{K-1})}_{L_{f_\theta} h(\boldsymbol{x})} + \alpha_1(h(\boldsymbol{x})) \geq 0.$$

Where $f_\theta(\boldsymbol{x}) = \theta^{\mathcal{P}}_{K-1,K} \boldsymbol{z}^{\mathcal{P}}_{K-1} + \theta^{\mathcal{N}}_{K-1,K} \boldsymbol{z}^{\mathcal{N}}_{K-1}$. Combining the last two equations, we have

$$\Psi(\theta^{\mathcal{P}}_{K-1,K}|\boldsymbol{x}) = \frac{dh(\boldsymbol{x})}{d\boldsymbol{x}} \dot{\boldsymbol{x}} + \alpha_1(h(\boldsymbol{x})) \geq 0.$$

which is equivalent to

$$\psi_1(\boldsymbol{x}, \theta^{\mathcal{P}}_{K-1,K}) = \dot{h}(\boldsymbol{x}) + \alpha_1(h(\boldsymbol{x})) \geq 0,$$

Since α_1 is a class $\mathcal{K}$ function,

$$\alpha_1(h(\boldsymbol{x})) \to 0 \text{ as } h(\boldsymbol{x}) \to 0.$$

In other words, we have $\dot{h}(\boldsymbol{x}) \geq 0$ when $h(\boldsymbol{x}) = 0$. Then, by Nagumo's theorem, we have that $h(\boldsymbol{x}) \geq 0$ is satisfied if $h(\boldsymbol{x}(t_0)) \geq 0$ since the derivative of $h(\boldsymbol{x})$ is non-decreasing on the hyperplane $h(\boldsymbol{x}) = 0$. In other words, we have that

$$h(\boldsymbol{x}(t)) \geq 0, \forall t \geq 0,$$

and the neural ODE is invariant. ■

Existence of class $\mathcal{K}$ function α_1 in Theorem 3.3:

Given a continuously differentiable $h(\boldsymbol{x}) \geq 0$. If $h(\boldsymbol{x}(t_0)) \geq 0$, there always exists such a class $\mathcal{K}$ function α_1 in (10) that allows us to freely choose $\theta^{\mathcal{P}}_{K-1,K}$ to make (10) satisfied as $\theta^{\mathcal{P}}_{K-1,K}$ is usually unconstrained. ■

A.2. Proof of Theorem 3.4 (Invariance Propagation to Nonlienar Layers)

The auxiliary dynamics are defined as

$$\dot{\theta}^{\mathcal{P}}_k = A^{\mathcal{P}}_k \theta^{\mathcal{P}}_k + B^{\mathcal{P}}_k u^{\mathcal{P}}_k$$

The condition in the theorem is given as

$$\Psi(u^{\mathcal{P}}_k|\boldsymbol{x}) = \underbrace{\frac{d^2 h(\boldsymbol{x})}{d\boldsymbol{x}^2} f^2_\theta(\boldsymbol{x}) + (\frac{dh(\boldsymbol{x})}{d\boldsymbol{x}} \frac{\partial f_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}} + \frac{d\alpha_1(h(\boldsymbol{x}))}{d\boldsymbol{x}}) f_\theta(\boldsymbol{x})}_{L^2_{f_\theta} h(\boldsymbol{x}) + L_{f_\theta}(\alpha_1 \circ h)(\boldsymbol{x})} + \underbrace{\frac{dh(\boldsymbol{x})}{d\boldsymbol{x}} \frac{\partial f_\theta(\boldsymbol{x})}{\partial \theta^{\mathcal{P}}_k} (A^{\mathcal{P}}_k \theta^{\mathcal{P}}_k + B^{\mathcal{P}}_k u^{\mathcal{P}}_k)}_{L_{g_u} L_{f_\theta} h(\boldsymbol{x})} + \alpha_2(\psi_1(\boldsymbol{x})) \geq 0,$$

Where $g_u = A^{\mathcal{P}}_k \theta^{\mathcal{P}}_k + B^{\mathcal{P}}_k u^{\mathcal{P}}_k$. Combining the last two equations, we have

$$\Psi(u^{\mathcal{P}}_k|\boldsymbol{x}) = \frac{d^2 h(\boldsymbol{x})}{d\boldsymbol{x}^2} f^2_\theta(\boldsymbol{x}) + \frac{dh(\boldsymbol{x})}{d\boldsymbol{x}} \dot{f}_\theta(\boldsymbol{x}) + \frac{d\alpha_1(h(\boldsymbol{x}))}{dt} + \alpha_2(\psi_1(\boldsymbol{x})) \geq 0,$$

Since $\psi_1(\boldsymbol{x}) = \dot{h}(\boldsymbol{x}) + \alpha_1(h(\boldsymbol{x}))$, the above equation is equivalent to

$$\psi_2(\boldsymbol{x}) := \dot{\psi}_1(\boldsymbol{x}) + \alpha_2(\psi_1(\boldsymbol{x})) \geq 0,$$

where $\dot{\psi}_1$ is involved with the derivative of θ_k^p that is defined by the auxiliary dynamics. Since $\boldsymbol{x}(t_0)$ is such that $\psi_1(\boldsymbol{x}(t_0)) \geq 0$, then by Theorem. 3.3, we have that

$$\psi_1(\boldsymbol{x}(t)) \geq 0, \forall t \geq t_0.$$

Further,

$$\psi_1(\boldsymbol{x}) = \dot{h}(\boldsymbol{x}) + \alpha_1(h(\boldsymbol{x})) \geq 0,$$

following (9). Again by Theorem. 3.3, since $h(\boldsymbol{x}(t_0)) \geq 0$, we have that

$$h(\boldsymbol{x}(t)) \geq 0, \forall t \geq t_0,$$

and thus the neural ODE (1) is invariant. ■

Using a similar technique as in the proof of Theorem 3.4, the proposed invariance is provably correct for stacked neural ODEs that may introduce higher relative degree $h(\boldsymbol{x})$.

Existence of class $\mathcal{K}$ functions α_1, α_2 in Theorem 3.3:

Given a continuously differentiable $h(\boldsymbol{x}) \geq 0$. If $h(\boldsymbol{x}(t_0)) > 0$, there always exists such a class $\mathcal{K}$ function α_1 such that $\psi_1(\boldsymbol{x}(t_0)) > 0$ as $\psi_1(\boldsymbol{x}) = \dot{h}(\boldsymbol{x}) + \alpha_1(h(\boldsymbol{x}))$. Then, we can always find a class $\mathcal{K}$ function α_2 in (13) that allows us to freely choose u_k^p to make (13) satisfied as u_k^p is usually unconstrained.

On the other hand, if $h(\boldsymbol{x}(t_0)) = 0$ and $\dot{h}(\boldsymbol{x}(t_0)) \geq 0$, we can also find such class $\mathcal{K}$ functions α_1, α_2 in (13) following similar analysis.

Finally, if $h(\boldsymbol{x}(t_0)) = 0$ and $\dot{h}(\boldsymbol{x}(t_0)) < 0$, then the specification will be immediately violated following Nagumo's theorem, and there is no way to enforce invariance. ■

A.3. Proof of Stability of Auxiliary Systems

The auxiliary dynamics are defined as

$$\dot{\theta}_k^{\mathcal{P}} = A_k^{\mathcal{P}} \theta_k^{\mathcal{P}} + B_k^{\mathcal{P}} u_k^{\mathcal{P}}$$

The CLF constraint in the invariance enforcing algorithm in the nonlinear case (the relaxation variable $\delta_j = 0$ when the CLF constraint does not conflict with the invariance enforcing constraint, i.e., when the output (state) of the neural ODE is far from undesired set boundaries) is

$$\Phi(u_k^{\mathcal{P}} | \theta_{k_j}^{\mathcal{P}}) = \frac{dV(\theta_{k_j}^{\mathcal{P}})}{d\theta_{k_j}^{\mathcal{P}}} (A_{k_j}^{\mathcal{P}} \theta_k^{\mathcal{P}} + B_{k_j}^{\mathcal{P}} u_k^{\mathcal{P}}) + \epsilon_j V(\theta_{k_j}^{\mathcal{P}}) \leq 0, j \in \{1, \ldots, d_k^{\mathcal{P}}\},$$

Combining the last two equations, we have

$$\Phi(u_k^{\mathcal{P}} | \theta_{k_j}^{\mathcal{P}}) = \dot{V}(\theta_{k_j}^{\mathcal{P}}) + \epsilon_j V(\theta_{k_j}^{\mathcal{P}}) \leq 0, j \in \{1, \ldots, d_k^{\mathcal{P}}\}, \tag{19}$$

Suppose we have

$$\dot{V}(\theta_{k_j}^{\mathcal{P}}) + \epsilon_j V(\theta_{k_j}^{\mathcal{P}}) = 0,$$

the solution to the above equation is

$$V(\theta_{k_j}^{\mathcal{P}}(t)) = V(\theta_{k_j}^{\mathcal{P}}(t_0)) e^{-\epsilon_j (t - t_0)},$$

Using the comparison lemma (Khalil, 2002), equation (19) implies that

$$V(\theta_{k_j}^{\mathcal{P}}(t)) \leq V(\theta_{k_j}^{\mathcal{P}}(t_0)) e^{-\epsilon_j (t - t_0)}, j \in \{1, \ldots, d_k^{\mathcal{P}}\},$$

Therefore,

$$V(\theta_{k_j}^{\mathcal{P}}(t)) \to 0, \text{ as } t \to \infty, \forall j \in \{1, \ldots, d_k^{\mathcal{P}}\},$$

and $\theta_{k_j}^{\mathcal{P}}$ is exponentially stabilized to $\theta_{k_j}^{\mathcal{P}\dagger}$ as $V(\theta_{k_j}^{\mathcal{P}}) = (\theta_{k_j}^{\mathcal{P}} - \theta_{k_j}^{\mathcal{P}\dagger})^2$. ■

B. Discussion on Auxiliary Dynamics

The auxiliary dynamics are defined as

$$\dot{\theta}_k^{\mathcal{P}} = A_k^{\mathcal{P}} \theta_k^{\mathcal{P}} + B_k^{\mathcal{P}} u_k^{\mathcal{P}} \tag{20}$$

where the above system is controllable if $A_k^{\mathcal{P}}$ and $B_k^{\mathcal{P}}$ are chosen such that

$$\left[B_k^{\mathcal{P}}, A_k^{\mathcal{P}} B_k^{\mathcal{P}}, \ldots, (A_k^{\mathcal{P}})^{m-1} B_k^{\mathcal{P}}\right]$$

is in full rank, where $m \in \mathbb{N}$ is a finite time step that drives the system from an initial state to a final state.

The exact choice of $A_k^{\mathcal{P}}$ and $B_k^{\mathcal{P}}$ may affect the performance. In other words, they will determine how close the output trajectory of the neural ODE can stay from the boundaries of undesired sets, and they will also determine how the parameters vary. From our experience, the effect on performance is pretty small since both the auxiliary control $u_k^{\mathcal{P}}$ and the parameter $\theta_k^{\mathcal{P}}$ are unbounded in the neural ODE, and thus, we can always quickly change the parameter $\theta_k^{\mathcal{P}}$ under any $A_k^{\mathcal{P}}$ and $B_k^{\mathcal{P}}$ that make the auxiliary system controllable. For simplicity, we can choose $A_k^{\mathcal{P}}$ and $B_k^{\mathcal{P}}$ to be zero and identity matrices, respectively.

On the other hand, we can make $A_k^{\mathcal{P}}$ and $B_k^{\mathcal{P}}$ be trainable parameters in our QP implementation via differentiable QP (Amos & Kolter, 2017).

C. Training neural ODEs with Invariance

The main objective is to make sure neural ODEs satisfy specifications during training, which also allows us to train class $\mathcal{K}$ functions in Theorems 3.3 and 3.4.

For neural ODEs that enforce the invariance on the input $\mathbf{I}$, the training is performed via the standard stochastic gradient descent. While training neural ODEs for enforcing the invariance on the model parameter θ, it is challenging to train both the neural ODE and differentiable QP simultaneously in the same pipeline. Thus, we propose the following two-stage training method: In the first stage, we train the neural ODE as usual and thus optimize the weight $\theta^\dagger$ of the network. In the second stage, we train the differentiable QP (specifically, the parameter in class $\mathcal{K}$ functions in QP (16)) such that θ minimally deviates from $\theta^\dagger$. The training of the network and the QP can be performed alternatively. We summarize the training process in Fig. 5.

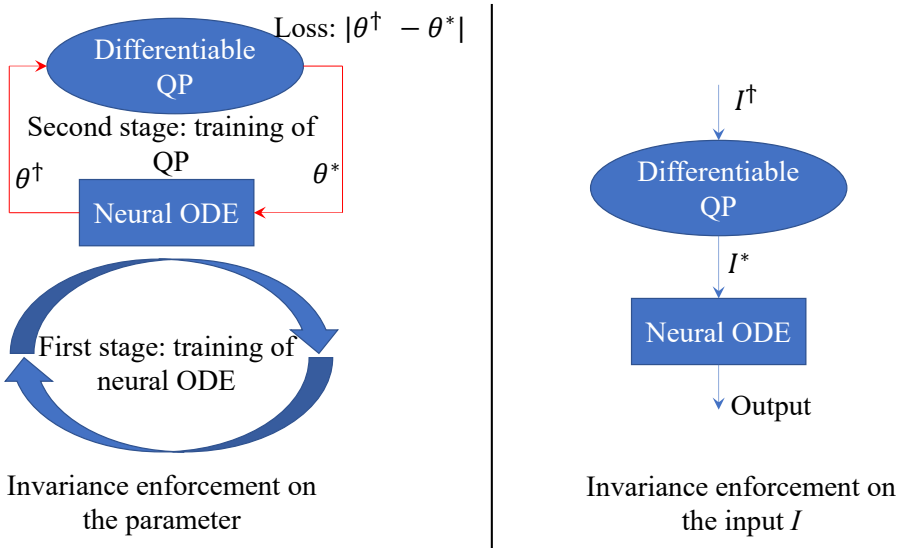

Figure 5. Training neural ODEs with invariance. The training of a neural ODE with $\mathbf{I}$ is regular when we enforce the invariance on $\mathbf{I}$.

D. Complex Specifications

Although the invariance of neural ODEs can be applied to a wide class of problems, one of the important applications is in the safe control of dynamical systems, as this involves complex output specifications. Consider the case where the output of the neural ODE controller is directly taken as the control of the dynamical system. The dynamical system is usually required to satisfy some safety constraints that are defined over its state instead of over its control. In other words, the specification of the neural ODE is not directly on its output. To map a state constraint onto the control of a dynamics system (i.e., the output of the neural ODE), we can use the CBF method.

More specifically, consider a control system whose dynamics are defined in the form:

$$\dot{\boldsymbol{y}} = f(\boldsymbol{y}, \boldsymbol{u}), \tag{21}$$

where $\boldsymbol{y} \in \mathbb{R}^q$ is the state of the system, $\boldsymbol{u} = \boldsymbol{x}$ is its control (i.e., the output of the neural ODE). $f : \mathbb{R}^{q \times n_1} \to \mathbb{R}^q$, where n_1 is the dimension of x_1 (or the control).

We wish the state of the system (21) to satisfy the following (safety) constraint:

$$b(\boldsymbol{y}) \geq 0, \tag{22}$$

where $b : \mathbb{R}^q \to \mathbb{R}$ is continuously differentiable, and its relative degree with respect to $\boldsymbol{u}$ is $d \in \mathbb{N}$.

As shown in (Xiao & Belta, 2022), we can use a HOCBF to enforce the safety constraint (22) for system (21). In other words, we map the safety constraint (22) onto the following HOCBF constraint:

$$\frac{d\phi_{d-1}}{d\boldsymbol{y}} f(\boldsymbol{y}, \boldsymbol{u}) + \alpha_d(\phi_{d-1}(\boldsymbol{y})) \geq 0, \tag{23}$$

where $\phi_k(\boldsymbol{y}) = \dot{\phi}_{k-1}(\boldsymbol{y}) + \alpha_k(\phi_{k-1}(\boldsymbol{y})), k = \{1, \ldots, d-1\}$ and $\phi_0(\boldsymbol{y}) = b(\boldsymbol{y})$, $\alpha_k, k \in \{1, \ldots, d\}$ are class $\mathcal{K}$ functions. It is worth noting that the construction of the HOCBF is similar to the construction (9) of the invariance of a neural ODE. The satisfaction of the above HOCBF constraint (23) implies the satisfaction of the safety constraint (22).

Since the output of the neural ODE is used to control the dynamical system, i.e., $\boldsymbol{u} = \boldsymbol{x}$, we can find the output constraint of the neural ODE from (23) in the form:

$$h(\boldsymbol{x}, \boldsymbol{y}) = \frac{d\phi_{d-1}}{d\boldsymbol{y}} f(\boldsymbol{y}, \boldsymbol{x}) + \alpha_d(\phi_{d-1}(\boldsymbol{y})) \geq 0, \tag{24}$$

Then, we can back-propagate the invariance of the neural ODE (i.e., $h(\boldsymbol{x}(t), \boldsymbol{y}(t)) \geq 0, \forall t \geq 0$) to the input $\mathbf{I}$ or its parameter, as shown before. In fact, in this scenario, the neural ODE serves as an integral controller for dynamical systems. In the original CBF method, we need to assume that the dynamics (21) are in affine control form in order to use the CBF-based QP to efficiently find a safe controller. With the proposed method, such restriction (assumption) is removed. This shows the advantage of the invariance of neural ODEs in safety-critical control problems.

E. Stacked Neural ODEs

When we have stacked neural ODEs, the IP is similar. Suppose we wish to enforce the invariance on the k_{th} stacked neural ODE, then we may choose the k_{th} stacked neural ODE to have linear layers, and we can get a similar equation as (8). The difference is that we may need to define higher-order CBFs. The performance of stacked neural ODEs needs to be further studied before applying the proposed invariance. Note that the index of stacked neural ODEs is in the reverse order as the layer index in non-stacked neural ODEs to make a difference between them.

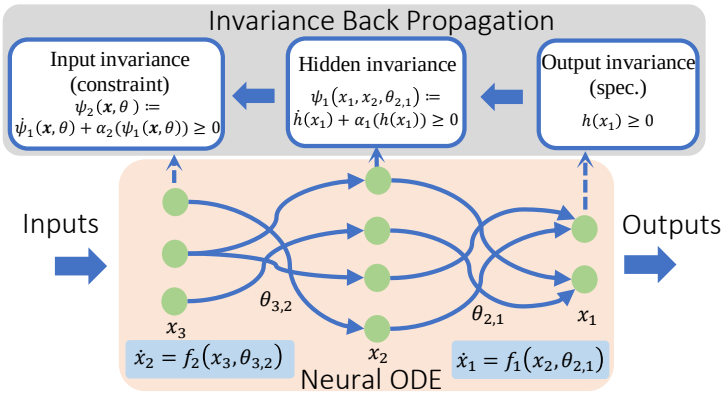

Figure 6. Relative degree and invariance example of three stacked neural ODEs. Recurrence is allowed, i.e., f_1, f_2 (neural networks) could be a function of x_1, x_2, respectively. x_3 is the input of the stacked neural ODE. There is an *odeint* after each layer.

For example, in Figure 6, the outputs of the neurons in x_3 are the inputs of the neurons in x_2. The neurons in x_3 are with one relative degree higher than the ones in x_2. The same applies to the neurons of x_2 and x_1. The highest relative degree of the three-layer neural ODE is two since the output neurons x_1 are defined to be with a relative degree of 0. The output specification $h(\boldsymbol{x}) \geq 0$ of a neural ODE can then be rewritten as $h(x_1) \geq 0$, as x_1 is the vector of output neurons. In order to show the relationships between the invariances of different layers of a neural ODE, we define the first-from-the-last hidden layer invariance $\psi_1 \geq 0$ (defined similarly as in Definition 3.1) as a function of $h(x_1)$ and its derivative, where ψ_1 is defined as:

$$\psi_1(x_1, x_2, \theta_{2,1}) := \dot{h}(x_1) + \alpha_1(h(x_1)), \tag{25}$$

where $\theta_{2,1}$ is the connection weight between layers 2 and 1, $x_2, \theta_{2,1}$ shows up in $\dot{h}(x_1)$, and $\alpha_1(\cdot)$ is a class $\mathcal{K}$ function (a class $\mathcal{K}$ is a strictly increasing function that passes through the origin). This way, the hidden invariance is related to the neurons x_1, x_2 and their connection weight $\theta_{2,1}$. We can define the invariance of any hidden (or input) layer k by functions $\psi_{k-1}(x_1, \ldots, x_k, \theta_{2,1}, \ldots, \theta_{k,k-1}) \geq 0, k \in \{2, \ldots, m+1\}$, recursively:

$$\begin{aligned} \psi_{k-1}(x_1, \ldots, x_k, \theta_{2,1}, \ldots, \theta_{k,k-1}) &:= \dot{\psi}_{k-2}(x_1, \ldots, x_{k-1}, \theta_{2,1}, \ldots, \theta_{k-1,k-2}) \\ &+ \alpha_{k-1}(\psi_{k-2}(x_1, \ldots, x_{k-1}, \theta_{2,1}, \ldots, \theta_{k-1,k-2})), \quad k \in \{1, \ldots, m+1\}, \end{aligned} \tag{26}$$

where $\alpha_{k-1}, k \in \{2, \ldots, m+1\}$ are class $\mathcal{K}$ functions, and $\psi_0(x_1, \theta_{1,0}) = h(x_1)$. $\theta_{k,k-1}$ denotes the connection weight between layers k and $k-1$. For the input layer $m+1$, we have $\psi_m(\boldsymbol{x}, \theta) = \psi_m(x_1, \ldots, x_{m+1}, \theta_{2,1}, \ldots, \theta_{m+1,m})$, where $\theta = (\theta_{2,1}, \ldots, \theta_{m+1,m})$, and the invariance of the input layer is illustrated by the input constraint $\psi_m(\boldsymbol{x}, \theta) \geq 0$. For example, the highest relative degree of the three stacked neural ODEs in Figure 6 is 2, and the input constraint (invariance) is $\psi_2(x_1, x_2, x_3, \theta_{2,1}, \theta_{3,2}) \geq 0$. Again, the performance of stacked neural ODEs should be studied first compared to non-stacked neural ODEs, we leave this for future work.

Order of HOCBFs. The order of HOCBFs in IP equals the relative degree of the specification (if it is from a dynamic system) plus the number of stacked neural ODEs, and plus one (if enforced on a nonlinear layer).

F. Experiment Details

In this section, we provide detailed settings for all the experiments, including some additional figures and results.

F.1. Spiral Curve Regression with Specifications

In this case, we enforce invariance on parameters in both the hidden nonlinear layer and the output linear layer.

Training data generation. The initial condition for the ODE we sample the data from is $[2, 0]$, and we sampled 1000 data points within the time interval [0,25] as the training data set. In order to make sure that the sampled data avoids the two critical regions in the case of invariance-in-training, we use CBFs to minimally modify the ODE. In other words, the components $A[0, 1], A[1, 0]$ of the A matrix in the considered ODE are minimally changed by the following quadratic program:

$$\min_{a_1,a_2}(a_1 - A[0, 1])^2 + (a_2 - A[1, 0])^2 \tag{27}$$

s.t. CBF constraints:

$$\begin{aligned} &2(A[0, 1] - O_{x,1})A[1, 0]^3 a_1 + 2(A[1, 0] - O_{y,1})A[0, 1]^3 a_2 + h_1(x) \geq 0, \\ &2(A[0, 1] - O_{x,2})A[1, 0]^3 a_1 + 2(A[1, 0] - O_{y,2})A[0, 1]^3 a_2 + h_2(x) \geq 0, \end{aligned} \tag{28}$$

where $h_i(x) = (A[0, 1] - O_{x,i})^2 + (A[1, 0] - O_{y,i})^2 - R_i^2, i \in \{1, 2\}$, and $(o_{x,i}, o_{y,i})$ denotes the location of the undesired set i, R_i denotes its size ($R_1 = R_2 = 0.2$ in the experiments).

After solving the above QP at each time and obtaining a_1^*, a_2^*, we replace $A[0, 1], A[1, 0]$ by a_1^*, a_2^*, respectively, in the ODE (please find details in the attached code).

Model structure. The training implementation and the enforcing QP for the invariance of the neural ODE are also given in the attached code. The f_θ in the neural ODE (1) is a three-layer fully connected network with sizes 2, 50, and 2, respectively. The activation functions used in the hidden layers are tanhshrink, while the output layers are without activation functions (linear layers).

Training. The training epoch is 500, and the training batch size is 20 with a batch sequence time of 10. We use RMSprop optimizer with learning rate $1e^{-3}$. The training time is about 2 hours on an RTX3090 GPU.

Invariance enforcing on the output linear layer. In this case, the neural ODE can be rewritten in linear form as in (8) in terms of the output layer parameters. Therefore, the invariance implementation can be achieved by directly changing the output layer parameters.

The specifications are defined as $h_i(\boldsymbol{x}) = (x - O_{x,i})^2 + (y - O_{y,i})^2 - R_i^2, i \in \{1, 2\}$. Therefore, in (10), we have

$$\frac{dh_i(\boldsymbol{x})}{d\boldsymbol{x}} = [2(x - O_{x,i}), 2(y - O_{y,i})],$$

and we choose the class $\mathcal{K}$ function α_1 as a linear function.

The QP (16) in this case is

$$\theta^{\mathcal{P}*}_{K-1,K} = \arg\min_{\theta^{\mathcal{P}}_{K-1,K}} ||\theta^{\mathcal{P}}_{K-1,K} - \theta^{\mathcal{P}\dagger}_{K-1,K}||^2, \tag{29}$$

s.t.

$$\frac{dh_1}{d\boldsymbol{x}}\theta^{\mathcal{P}}_{K-1,K}\boldsymbol{z}^{\mathcal{P}}_{K-1} + \frac{dh_1}{d\boldsymbol{x}}\theta^{\mathcal{N}}_{K-1,K}\boldsymbol{z}^{\mathcal{N}}_{K-1} + k_1 h_1(\boldsymbol{x}) \geq 0, \tag{30}$$

$$\frac{dh_2}{d\boldsymbol{x}}\theta^{\mathcal{P}}_{K-1,K}\boldsymbol{z}^{\mathcal{P}}_{K-1} + \frac{dh_2}{d\boldsymbol{x}}\theta^{\mathcal{N}}_{K-1,K}\boldsymbol{z}^{\mathcal{N}}_{K-1} + k_2 h_2(\boldsymbol{x}) \geq 0, \tag{31}$$

where $k1 = k2 = 10$ when the invariance is enforced after the training of the neural ODE.

Invariance enforcing on the hidden nonlinear layer. The neural ODE (1) can be explicitly written as

$$\dot{\boldsymbol{x}} = f_\theta(\boldsymbol{x}) = \theta_{2,3}(tanhshrink(\theta_{1,2}\boldsymbol{x}^3 + b_{1,2})) + b_{2,3}, \tag{32}$$

The auxiliary dynamics (11) is defined as

$$\dot{\theta}^{\mathcal{P}}_1 = u^{\mathcal{P}}_1, \tag{33}$$

where $\theta_1^{\mathcal{P}}$ is the vector form of $\theta_{1,2}^{\mathcal{P}}$, and $\theta_{1,2}^{\mathcal{P}}$ is the partial parameter of $\theta_{1,2}$ in (32) that we wish to propagate the invariances to.

Therefore, in (13), we have

$$\frac{dh_i(\boldsymbol{x})}{d\boldsymbol{x}} = [2(x - O_{x,i}), 2(y - O_{y,i})], i \in \{1, 2\}$$

$$\frac{\partial f_\theta(\boldsymbol{x})}{\partial \theta_1^{\mathcal{P}}} = \theta_{2,3} tanh^2(\theta_{1,2}\boldsymbol{x}^3 + b_{1,2})\dot{\theta}_{1,2}\boldsymbol{x}^3 (\text{part of } \dot{\theta}_{1,2} \text{ i.e., } \dot{\theta}_1^{\mathcal{P}}, \text{ is defined as } u_1^{\mathcal{P}} \text{ following (33)}),$$

$$\frac{\partial f_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}} = 3\theta_{2,3} tanh^2(\theta_{1,2}\boldsymbol{x}^3 + b_{1,2})\theta_{1,2}\boldsymbol{x}^2 f_\theta(\boldsymbol{x}),$$

$$\frac{dh_i^2(\boldsymbol{x})}{d\boldsymbol{x}^2} = [2, 2], i \in \{1, 2\},$$

and we choose class $\mathcal{K}$ functions α_1, α_2 as linear functions. The CLFs used to stabilize $\theta_1^{\mathcal{P}}$ is defined as (17).

The QP (18) in this case is

$$(u_1^{\mathcal{P}*}, \delta_{1:d_1^{\mathcal{P}}}^*) = \arg\min_{u_1^{\mathcal{P}}, \delta_{1:d_1^{\mathcal{P}}}} ||u_1^{\mathcal{P}}||^2 + \sum_{j=1}^{d_1^{\mathcal{P}}} w_j \delta_j^2, \tag{34}$$

s.t.

$$\frac{d^2 h_1(\boldsymbol{x})}{d\boldsymbol{x}^2} f_\theta^2(\boldsymbol{x}) + \frac{dh_1(\boldsymbol{x})}{d\boldsymbol{x}} \frac{\partial f_\theta(\boldsymbol{x})}{\partial \theta_1^{\mathcal{P}}} u_1^{\mathcal{P}} + (\frac{dh_1(\boldsymbol{x})}{d\boldsymbol{x}} \frac{\partial f_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}} + \frac{dk_1 h_1(\boldsymbol{x})}{d\boldsymbol{x}}) f_\theta(\boldsymbol{x}) + k_2 \psi_{1,1}(\boldsymbol{x}) \geq 0, \tag{35}$$

$$\frac{d^2 h_2(\boldsymbol{x})}{d\boldsymbol{x}^2} f_\theta^2(\boldsymbol{x}) + \frac{dh_2(\boldsymbol{x})}{d\boldsymbol{x}} \frac{\partial f_\theta(\boldsymbol{x})}{\partial \theta_1^{\mathcal{P}}} u_1^{\mathcal{P}} + (\frac{dh_2(\boldsymbol{x})}{d\boldsymbol{x}} \frac{\partial f_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}} + \frac{dk_1 h_2(\boldsymbol{x})}{d\boldsymbol{x}}) f_\theta(\boldsymbol{x}) + k_2 \psi_{1,2}(\boldsymbol{x}) \geq 0, \tag{36}$$

$$\frac{dV(\theta_{1_j}^{\mathcal{P}})}{d\theta_{1_j}^{\mathcal{P}}} u_1^{\mathcal{P}} + \epsilon_j V(\theta_{1_j}^{\mathcal{P}}) \leq \delta_j, j \in \{1, \ldots, d_1^{\mathcal{P}}\}, \tag{37}$$

where $k_1 = 20, k_2 = 100, \epsilon_j = 10, w_j = 1$ in Table 1. $\psi_{1,j}(\boldsymbol{x}) = \dot{h}_j(\boldsymbol{x}) + k_1 h_j(\boldsymbol{x}), j \in \{1, 2\}$.

Comparison between invariance enforcing on different layers. We present an illustrative example for the comparison between neural ODE and invariances enforced on different layers in Fig. 7. The chosen number of parameters is 6 for both the hidden invariance and output invariance. As expected, the output specifications are guaranteed to be satisfied in the invariances, while they are violated in the (pure) neural ODE. We further illustrate the value of $h(\boldsymbol{x}) = \min\{h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\}$ in Fig. 8, in which $h(\boldsymbol{x}) \geq 0$ denotes the satisfaction of all the specifications.

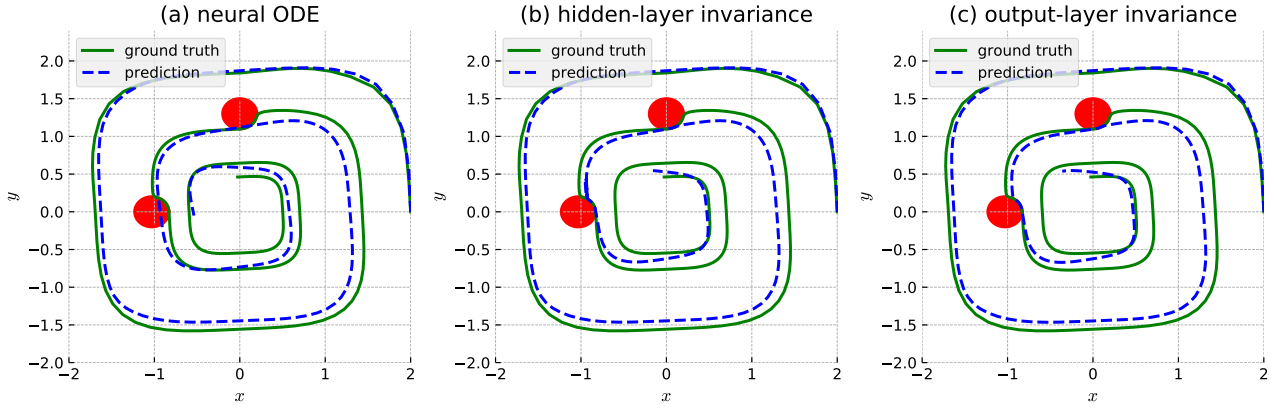

Figure 7. Spiral curve performance comparisons between neural ODE and invariance propagated to the hidden and output layers. The prediction of the neural ODE violates the output specifications.

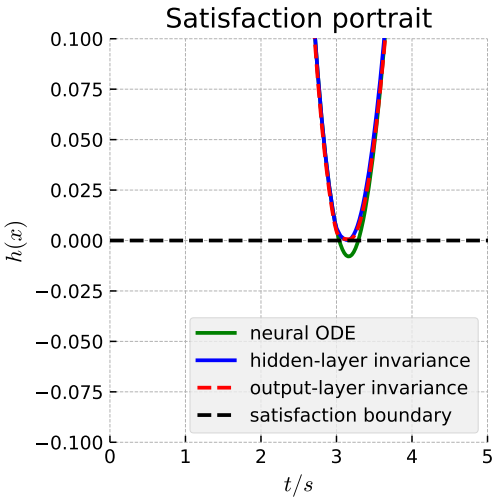

Figure 8. Spiral curve regression: specification satisfaction portraits. $h(\boldsymbol{x}) = \min\{h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\} \geq 0$ denotes the satisfaction of all the specifications. $h_j(\boldsymbol{x}) = (x - x_{o_j})^2 + (y - y_{o_j})^2 - R^2, j \in \{1, 2\}$.

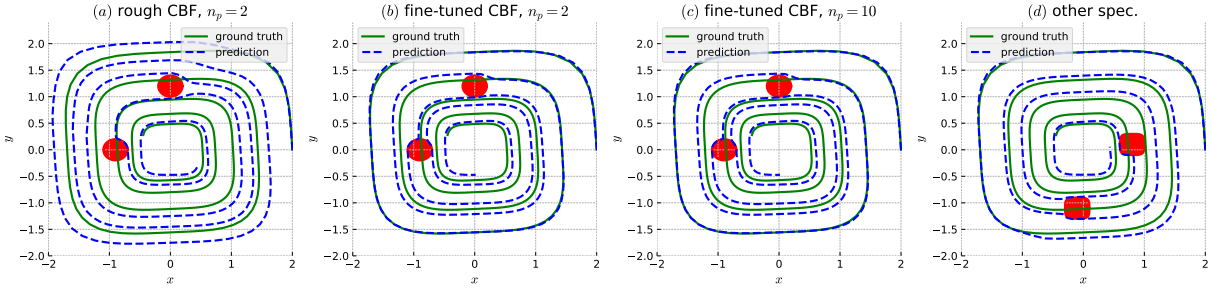

Figure 9. After-training invariance enforcing for spiral curve regression with output specifications. n_p denotes the number of parameters randomly chosen in the QP (16). (a)-(c) are with two circular undesired sets defined by: $h_j(\boldsymbol{x}) = (x - x_{o_j})^2 + (y - y_{o_j})^2 - R^2$, (x_{o_j}, y_{o_j}) denotes the location of undesired set $j \in S, R > 0$, and (d) is with two randomly-placed superellipse-type undesired sets defined by: $h_j(\boldsymbol{x}) = (x - x_{o_j})^4 + (y - y_{o_j})^4 - R^4$.

After training. In this case, since the neural ODE does not have the external input $\mathbf{I}$, we minimally change the parameters of the model to enforce the invariance using the proposed QP-based approach (16). As a result, the outputs of the neural ODE can satisfy all the specifications (see Figures 9 (a)-(d)). The model is already trained, we need to carefully choose the class $\mathcal{K}$ functions shown in (9). Otherwise, the resulting trajectory would be overly-conservative such that there is a large deviation from the original one even when the state is far away from the constraint boundary $h_j(\boldsymbol{x}) = 0, j \in S$, as shown in Fig. 9a. With slightly fined-tuned CBF parameters p_i (i.e., its class $\mathcal{K}$ functions), the conservativeness can be addressed, as shown in Figure 9b.

We can also minimally change a different number of parameters of the neural ODE. Comparing Fig. 9b with Figure 9c, the outputs of the neural ODE are almost the same under 2 or 10 randomly chosen model parameters. This demonstrates the effectiveness of the proposed invariance. Although the computation efficiency is better when we choose to modify fewer weights using the proposed QP-based approach (16), the performance might actually be worse when the learning task is complicated as we have to largely change the parameters. In order to show the robustness of the proposed invariance, we also tested other types of specifications, as the two randomly-placed superellipse-type undesired sets shown in Figure 9d. The outputs of the neural ODE can also guarantee the satisfaction of the corresponding output specifications. Tuning the CBF parameters in the invariance may be non-trivial when there are many specifications. Thus, we show next how we can enforce invariance in the training loop.

F.2. Convexity Portrait of a Function

In this case, we enforce invariance on parameters in both the hidden nonlinear layer and the output linear layer.

Training data generation. The convex function we consider for sampling the training data is $g(x) = x^2$, and we set $\mu_1 = \mu_2 = 0.5$ for the Jensen's inequality. $x = t, y = t + 2 - \frac{1.9}{10}t$, where $t \in [0, 10]$. We sampled 100 data points as the training data set. The CBF $h(x, y)$ for Jensen's inequality in the neural ODE is defined as:

$$h(x, y) = \mu_1 z_1 + \mu_2 z_2 - z_3, \tag{38}$$

where z_1, z_2, z_3 denote the three outputs of the neural ODE. The implementation details and the enforcing QP for the invariance are given in the attached code.

Model structure. The f_θ in the neural ODE (1) is a three-layer fully connected network with sizes 3, 50, and 3, respectively. The activation functions used in the model are tanh.

Training. The training epoch is 2000, and the training batch size is 20 with a batch sequence time of 10. We use RMSprop optimizer with learning rate $1e^{-3}$. The training time is about 1 hour on an RTX3090 GPU.

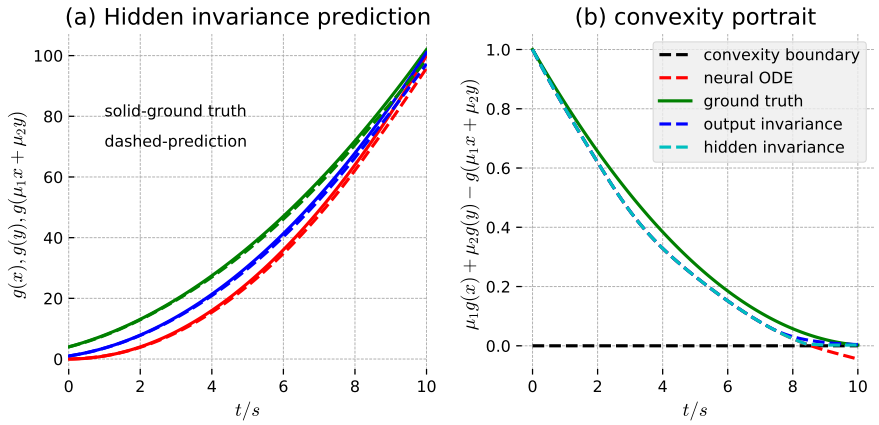

Figure 10. Convexity portrait of the neural ODE outputs. The non-negativity of the functions in (b) demonstrates the satisfaction of Jensen's inequality.

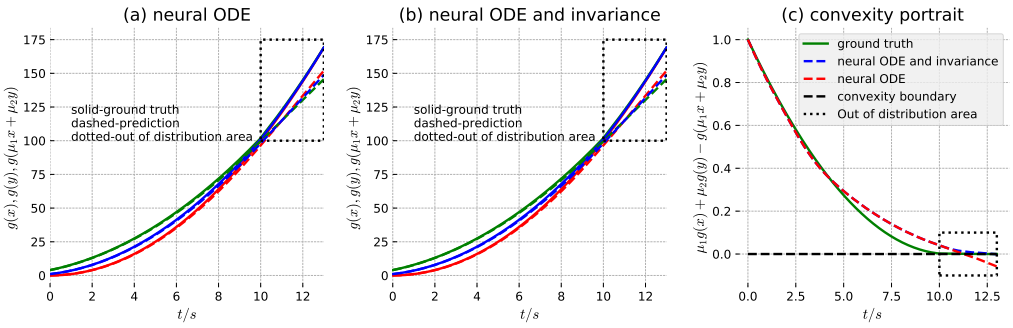

Figure 11. Convexity portrait of the neural ODE outputs. x, y are functions of t. The non-negativity of the functions in (c) demonstrates the satisfaction of Jensen's inequality.

In distribution. Within the range of the training data, the trained neural ODE is not guaranteed to satisfy Jensen's inequality as illustrated by the red-dashed curve in Figure 10b. However, with the proposed (hidden and output) invariances, the model outputs are guaranteed to satisfy the Jensen's inequality, as shown by the blue-dashed and cyan dashed curves in Figure 10b.

Out of distribution. Although the outputs of a trained neural ODE model may satisfy Jensen's inequality within the range of the training data set, they may still violate Jensen's inequality when we conduct predictions for future time (out of the

training data range), as shown by the red-dashed curve in Figure 11c. However, with the proposed invariance, we can guarantee that the future prediction of the model also satisfies Jensen's inequality (see blue dashed line in Fig. 11c).

F.3. HalfCheetah-v2 and Walker2d-v2 kinematic modeling

In this case, we enforce invariance on parameters in the output linear layer.

We evaluate our invariance framework on two publicly available datasets for modeling physical dynamical systems (Lechner & Hasani, 2022; Hasani et al., 2021). The two datasets consist of trajectories of the HalfCheetah-v2 and Walker2d-v2 3D robot systems (Brockman et al., 2016) generated by the Mujoco physics engine (Todorov et al., 2012). Each trajectory represents a sequence of a 17-dimensional vector describing the system's state, such as the robot's joint angles and poses. For each of the two tasks, we define 34 safety constraints that restrict the system's evolution to the value ranges observed in the dataset, i.e., the joint limitations.

Model structure. The f_θ in the neural ODE (1) is a three-layer fully connected network with sizes 17, 64, and 17, respectively. The activation functions used in the model are Tanh.

Training. The training epoch is 200, and the training batch size is 64 with a batch sequence time of 20. We use RMSprop optimizer with learning rate $1e^{-3}$. The training time is about 1 hour on an RTX3090 GPU.

F.4. Lidar-based End-to-End Autonomous Driving

In lidar-based driving, we assume the states of the ego and ado vehicles are obtained by other sensors (e.g. GPS or communication). We use the proposed invariance to back-propagate the safety requirements of the ego vehicle all the way to the input layer of the neural ODE, i.e., finding a constraint on the Lidar input I that can guarantee the safety of the ego vehicle.

Problem setup. The ego vehicle state $\boldsymbol{x} = (x, y, \theta, v)$ (along-lane location, off-center distance, heading, and speed, respectively) follows the unicycle vehicle dynamics, and the other vehicle moves at a constant speed. The ego vehicle is initially behind the other moving vehicle, and its *objective* is to overtake the other vehicle while avoiding collisions. The collision avoidance is characterized by a safety constraint $b(\boldsymbol{x}, \boldsymbol{x}_p) = (x - x_p)^2 + (y - (y_p + y_d))^2 - R^2 \geq 0$, where $\boldsymbol{x}_p \in \mathbb{R}^4$ denotes the state of the preceding vehicle, and $(x_p, y_p) \in \mathbb{R}^2$ denotes the location of the preceding vehicle. $y_d \in \mathbb{R}$ is the off-center distance of the covering disk with respect to the center of the other vehicle. The satisfaction of $b(\boldsymbol{x}, \boldsymbol{x}_p) \geq 0$ implies collision-free.

Training setup. In order to train a neural ODE controller that can be applied to the ego vehicle in closed-loop testing, we randomly assign locations for the ego vehicle with random states around the other vehicle. Then, we use a safety-guaranteed CBF-based QP controller to generate safe controls for the ego vehicle to overtake the other vehicle. We sampled 200 trajectories as the training data, and each trajectory has a time sequence of states and controls with a length of 100. In order to effectively train the neural ODE model, we also take the states of the ego and other vehicles as input to the neural ODE in addition to the Lidar information.

Training data generation. The training data comes from an integrated simulation environment (not released yet), and it is given as a "pickle" file. There are 200 randomly sampled trajectories and the corresponding safe controls coming from a CBF controller, and each trajectory is with 100 time-sequence of data with a discretization time of 0.1s. The Lidar information is given as a sequence of data with size 1x100, and each data point denotes a distance metric with respect to an obstacle from the angle 0 to 2π. The Lidar sensing range is 20m.

During training, we normalize the Lidar information by multiplying the data with a factor of 1/200. The ego vehicle speed is also normalized by multiplying the speed with a factor of 1/180 when it is taken as an external input. The normalization of the external input is to ensure that the neural ODE can converge during training.

Model structure. The neural ODE (1) is a five-layer fully connected network with sizes 2, 64, 256, 512, and 206, respectively. The activation functions used in the model are GELU. Since we enforce the invariance on the external lidar input, we reformulate the neural ODE (1) into:

$$\dot{\boldsymbol{x}} = f_\theta(\boldsymbol{x}) + g_\theta(\boldsymbol{x})\mathbf{I}, \tag{39}$$

When employing feature extractors for the invariance, we use a Convolutional Neural Network (CNN) whose shape is given as $[[1, 4, 5, 2, 1], [4, 8, 3, 2, 1], [8, 12, 3, 2, 0]]$, where there are three layers, and the parameters of each layer denote input channels, output channels, kernel size, stride, and padding, respectively. After the CNN, we use a max pooling in each output channel to reduce the feature size from 100 to 12.

Training. The training epoch is 200 (each epoch includes the sampling of each of the 200 trajectories), and the training batch size is 20 with a batch sequence time of 10. We use RMSprop optimizer with learning rate $1e^{-3}$. The training time is about 24 hours on an RTX3090 GPU.

Invariance v.s. safe filters v.s. pure neural ODE. The ego vehicle starts at a speed of $18m/s$, while the other vehicle moves at a constant speed of $13.5m/s$. In the case of noise-free Lidar sensing, the ego may avoid collision when it overtakes the other moving vehicle with the neural ODE controller. However, with noisy Lidar, the neural ODE controller may cause the ego vehicle to collide with the other moving vehicle during the overtaking process, as the red trajectory shown in Figure 13. The safety constraint $b(\boldsymbol{x}, \boldsymbol{x}_p)$ becomes negative (as the red curve shown in Fig. 12) when the ego approaches the other moving vehicle, which implies collision.

Using the proposed invariance, We map the safety requirement of the ego vehicle onto a constraint on the noisy Lidar input $\mathbf{I}$. The dimension of the Lidar information is 1×100, and thus, the dimension of the decision variable of the QP that enforces the invariance is also 100. Even so, the QP can still be efficiently solved as it is just a convex optimization. With the proposed invariance, we can slightly modify the noisy Lidar data such that the outputs (controls) can guarantee the safety of the ego vehicle, as the green trajectory shown in Figure 13. The modified Lidar information (through invariance) is

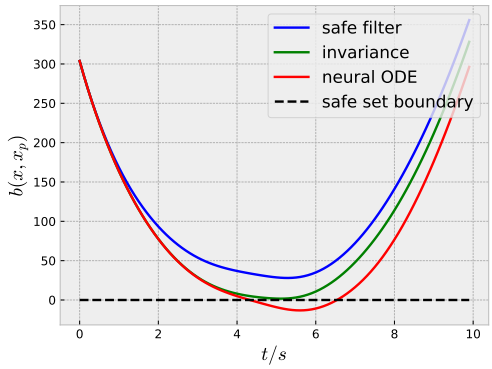

Figure 12. Comparison between safe filters, neural ODE, and invariance under a noisy Lidar point cloud. $b(\boldsymbol{x}, \boldsymbol{x}_p) \geq 0$ implies collision-free.

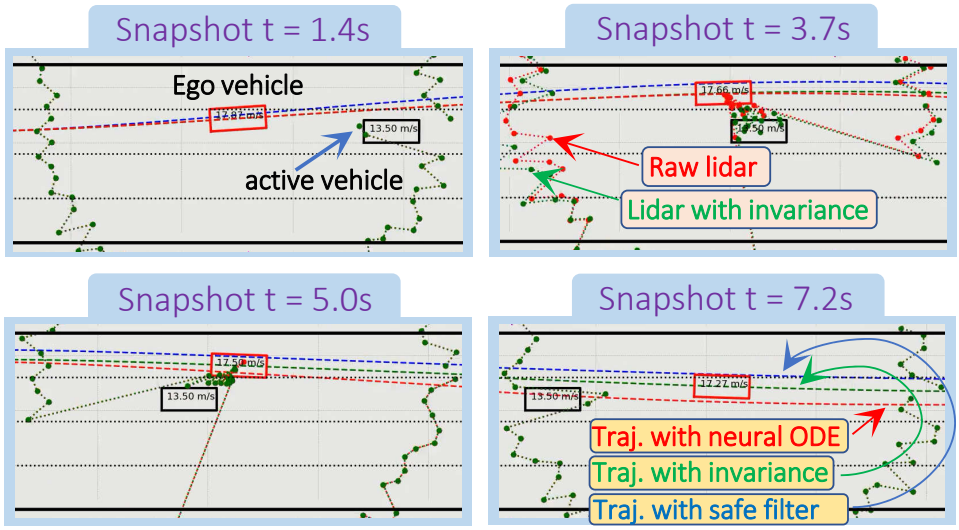

Figure 13. Snapshots of simulations with trajectory comparison between safe filters, neural ODE, and invariance under a noisy Lidar point cloud. $b(\boldsymbol{x}, \boldsymbol{x}_p) \geq 0$ implies collision-free. A safe filter may make the ego conservative, and thus the ego stays unnecessarily far away from the ground truth, while the pure neural ODE controller may cause collision under noise.

illustrated by the green-dotted curve in the snapshot $t = 3.7s$ of Figure 13.

Invariance with feature extractors. In cases where the inputs of the neural ODE have high dimensions, like Lidar-based control, we may use some neural networks (such as CNN) to reduce the dimension of input features, thus reducing the complexity of the QP that enforces the invariance. For the driving example, we used a CNN to reduce the 100-dimension Lidar information to 12-dimension features, and the results are similar to the case of raw Lidar. In other words, a collision may occur when with noisy Lidar input but can be guaranteed to avoid using the proposed invariance.