# On the Forward Invariance of Neural ODEs

**Anonymous authors**
Paper under double-blind review

## Abstract

To ensure robust and trustworthy decision-making, it is highly desirable to enforce constraints over a neural network's parameters and its inputs automatically by back-propagating output specifications. This way, we can guarantee that the network makes reliable decisions under perturbations. Here, we propose a new method for achieving a class of specification guarantees for neural Ordinary Differentiable Equations (ODEs) by using invariance set propagation. An invariance of a neural ODE is defined as an output specification, such as to satisfy mathematical formulae, physical laws, and system safety. We use control barrier functions to specify the invariance of a neural ODE on the output layer and propagate it back to the input layer. Through the invariance backpropagation, we map output specifications onto constraints on the neural ODE parameters or its input. The satisfaction of the corresponding constraints implies the satisfaction of output specifications. This allows us to achieve output specification guarantees by changing the input or parameters while maximally preserving the model performance. We demonstrate the invariance propagation on a comprehensive series of representation learning tasks, including spiral curve regression, autoregressive modeling of joint physical dynamics, convexity portrait of a function, and safe neural control of collision avoidance for autonomous vehicles.

## 1 Introduction

We wish to equip ODE-based networks with performance guarantees to enable decision-critical applications. To this end, we explore control-theoretic invariance. We observe that explicit and implicit data patterns are usually inherited from data that originates from real systems, such as robot sensors or virtual systems. These patterns can be described by specifications consisting of physical laws, mathematical expressions, safety constraints, and other prior knowledge of the structure of the data and the task. Guaranteeing task specifications for complex learning systems such as robot learning-based control is challenging due to the fact that the learned models perform representation learning in unstructured environments and are expected to generalize to unseen situations. Moreover, the learned models might be subjected to distribution shifts and cyber-physical attacks that could degrade the quality of their output decisions. We propose to propagate invariance through neural Ordinary Differentiable Equations (ODEs) (Chen et al., 2018) to equip them with performance guarantees. The key idea is to do invariance propagation for neural ODEs using High Order Control Barrier Function (HOCBF) theory (Xiao & Belta, 2022). However, the HOCBF method has significant limitations: $(a)$ it fails to work for system/data with unknown dynamics; $(b)$ the HOCBF method is conservative such that the performance could be low; $(c)$ it can only work efficiently for affine control systems, and it cannot be directly applied to complex neural networks. In this work, We address these limitations and generalize safety to any other system properties using the proposed invariance. Specifically, we use HOCBFs to represent the output specifications of neural ODEs as invariance and back-propagate the invariance from the output layer all the

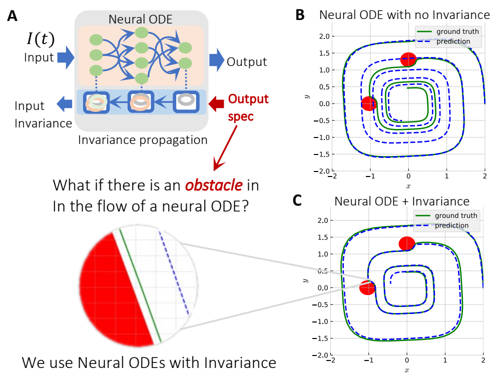


Figure 1: Invariance propagation for neural ODEs. Output specifications can be guaranteed with invariance, including specification satisfaction between samplings, e.g., spiral curve regression with critical region avoidance.

way back to the input layer or the parameters of the neural ODE. This process finds input/parameter constraints whose satisfactions imply the satisfactions of output specifications. Consider, for example, the spiral curve regression with output specifications, as shown in Fig. 1. The output of the neural ODE is trained to stay close to the ground truth. The output further is required to avoid the two "obstacles" denoted as the red regions marked in the path of the ground truth spiral curve. In general, it is hard to train a neural ODE to avoid obstacles placed at arbitrary locations, such as the two red regions (Dupont et al., 2019). Our proposed invariance back-propagates the output specification all the way to the input layer (or model parameters) and finds constraints on the input/parameters whose satisfaction implies the avoidance of these two critical regions.

**Contributions** $(i)$ we propose a new method we call invariance propagation (or, short, invariance) for neural ODEs that guarantees the satisfaction of output specifications; $(ii)$ we propose a quadratic program approach to enforce invariance on the input or parameters of neural ODEs; $(iii)$ we show how we can enforce complex specifications for dynamical systems with neural ODEs using invariance; $(iv)$ we address the possible conservativeness of the invariance by including invariance functions in the training loop; $(v)$ we illustrate the effectiveness of the invariance method on a series of temporal dynamics modeling tasks.

## 2 Related Works

**Neural ODEs.** 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 (Amini et al., 2020; Lechner & Hasani, 2020; Lechner et al., 2020; 2019; 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., 2021a). Recent methods provide safety guarantees for inference in a neural ODE system, e.g. stochastic reachability analysis (Gruenbacher et al., 2020; 2021). 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.** An invariant set has been widely used to characterize the safe behavior of dynamical systems (Preindl, 2016) (Rakovic et al., 2005) (Ames et al., 2014) (Glotfelter et al., 2017) Xiao & Belta (2019). 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). In (Ames et al., 2014), (Ames et al., 2020), (Glotfelter et al., 2017), it has been shown that a state constraint over a dynamical system can be mapped onto a control constraint using CBFs, and the satisfaction of the control constraint implies the satisfaction of the original state constraint.

**Filters for 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., 2021). 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), but OptNet is 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. They are offline methods, unable to adapt to environment changes (e.g., varying size of the unsafe sets) (Li, 2021). Optimizations as safe filters in neural networks could significantly limit the model performance as a filter may also discard useful features that are from all previous layers. In contrast, we propose to propagate the invariance (e.g. for safety) all the way to the input of the neural ODE. Thus, performance can be guaranteed by changing the input following the invariance. The invariance can also be applied to a wide class of performance guarantees in addition to safety.

## 3 Neural ODEs

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

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

where $\boldsymbol{x} \in \mathbb{R}^n$ is the neuron state, $NN_\theta$ is any neural network model parameterized by $\theta$. The output of $NN_\theta$ is also with dimension $n$, and the output of the neural ODE is the integration solution of (1).

Given an initial state $\boldsymbol{x}(t_0)$, we can train a neural ODE by the following optimization problem:

$$\theta^* = \arg\min_\theta \ell(\boldsymbol{x}^{obs}(t), \boldsymbol{x}^{pre}(t)), \tag{2}$$

where $\ell(\cdot,\cdot)$ is a similarity measurement, $\boldsymbol{x}^{obs}$ and $\boldsymbol{x}^{pre}$ denote the observed state and predicted state from the neural ODE (1), respectively. This optimization problem can then be solved and trained end-to-end by reverse mode automatic differentiation (Rumelhart et al., 1986; Pontryagin, 2018). Given a set of input observations $\mathbf{I}(t) \in \mathbb{R}^{n_a}$, where $n_a \in \mathbb{N}$, then the model is defined as follows:

$$\dot{\boldsymbol{x}}(t) = NN_\theta(\boldsymbol{x}(t), \mathbf{I}(t), t). \tag{3}$$

Neural ODEs equipped with inputs are especially useful for modeling temporal dynamics with neural agents deployed in control settings (Lechner et al., 2019; Hasani et al., 2020; Kidger et al., 2020; Lechner et al., 2020; Vorbach et al., 2021).

## 4 Method: Invariance of Neural ODEs

In this section, we specify the invariance of neural ODEs using the HOCBF theory (Xiao & Belta, 2022) while addressing its limitations $(a) - (c)$ as discussed in the introduction. We also generalize safety to other specifications. We omit the explicit dependence of $t$ in the neural ODEs (1) and (3) for simplicity as they can be handled similarly by time-varying HOCBFs defined in the same work. Suppose we have an output specification $h(\boldsymbol{x}) \geq 0$ for a neural ODE, where $h : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. Typical output specifications include system safety (such as collision avoidance in autonomous driving), physical laws (such as energy conservation), mathematical formulae (such as Jensen's inequality, Cauchy Schwarz inequality), etc. We assume neural ODEs are with differentiable activation functions. Our proposed method is to back-propagate the output specification to a constraint on the input or parameter of the neural ODE in order to tractably guarantee the satisfaction of output specifications. We first define the invariance of a neural ODE as follows.

**Definition** 1 *The invariance of a neural ODE is defined with respect to its output specification* $h(\boldsymbol{x}) \geq 0$ *such that if* $h(\boldsymbol{x}(0)) \geq 0$*, then* $h(\boldsymbol{x}(t)) \geq 0, \forall t \geq 0$.

Since the invariance of a neural ODE is defined in terms of its output, we may also call it the output invariance. If the neural ODE has multiple layers, we also have hidden invariances and input invariance, as shown next.

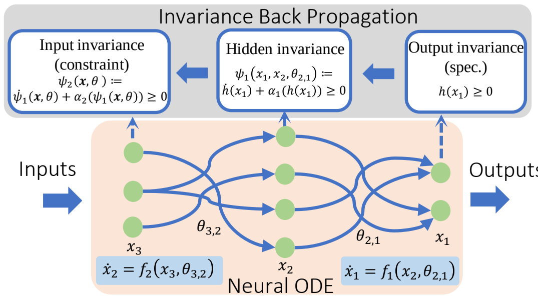

Invariance Back Propagation
Input invariance (constraint)
$\psi_2(\boldsymbol{x},\theta) := \dot\psi_1(\boldsymbol{x},\theta) + \alpha_2(\psi_1(\boldsymbol{x},\theta)) \geq 0$
Hidden invariance
$\psi_1(x_1,x_2,\theta_{2,1}) := \dot h(x_1) + \alpha_1(h(x_1)) \geq 0$
Output invariance (spec.)
$h(x_1) \geq 0$
Inputs
Outputs
$\theta_{3,2}$
$\theta_{2,1}$
$x_3$
$x_2$
$x_1$
$\dot x_2 = f_2(x_3,\theta_{3,2})$
$\dot x_1 = f_1(x_2,\theta_{2,1})$
Neural ODE

Figure 2: Relative degree and invariance example of a three-layer neural ODE. 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 neural ODE. There is an odeint after each layer.

**4.1 Invariance propagation for neural ODEs without external input I:** If the $NN_\theta$ in (1) is fully connected, then the relative degree of one neuron (suppose it is one of the outputs) with respect to another one is only one, as defined in nonlinear systems (Khalil, 2002). Otherwise, we have sparse neural ODEs (Lechner et al., 2020; Liebenwein et al., 2021) that induce high relative degrees. The method works as follows. The first step is to classify all the neurons of a neural ODE according to their relative degrees. This can be done according to the connection relationship between neurons. In a neural ODE, if the output of neuron $i$ is the input of neuron $j$ and not vice versa, then the relative degree of neuron $i$ is one relative degree higher than neuron $j$. The output neurons are defined to be with a relative degree of 0. Thus, we can denote the relative degree of a neuron in a neural ODE by $r \in \mathbb{N}$. We denote the neuron vectors as $x_r$ in which the relative degrees of all neuron components are $r - 1$. Then $\boldsymbol{x} = (x_1, \dots, x_{m+1})$, where $m$ denotes the highest relative degree of the neural ODE. Recurrent connections in a neural ODE will not change their relative degrees, i.e., the output of a neuron in the neuron vector $x_r$ could also be the input of any neuron in $x_r$.

For example, in Figure 2, 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 1) as a function of $h(x_1)$ and its derivative, where $\psi_1$ is defined as:

$$\psi_1(x_1, x_2, \theta_{2,1}) := \dot{h}(x_1) + \alpha_1(h(x_1)), \tag{4}$$

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:

$$\begin{aligned}\psi_{k-1}(x_1, \ldots, x_k, \theta_{2,1}, \ldots, \theta_{k,k-1}) &:= \dot{\psi}_{k-2}(x_1, \ldots, x_{k-1}, \theta_{2,1}, \ldots, \theta_{k-1,k-2}) \\ &+ \alpha_{k-1}(\psi_{k-2}(x_1, \ldots, x_{k-1}, \theta_{2,1}, \ldots, \theta_{k-1,k-2})), \quad k \in \{1, \ldots, m+1\},\end{aligned} \tag{5}$$

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-layer neural ODE in Figure 2 is 2, and the input constraint (invariance) is $\psi_2(x_1, x_2, x_3, \theta_{2,1}, \theta_{3,2}) \geq 0$. We have the following theorem to show the relationships among invariances in the neural ODE (1):

**Theorem** 1 *Given a neural ODE defined by (1), the output specification $h(x_1) \geq 0$, and the functions $\psi_{k-1}$ defined by (5), if there exist class $\mathcal{K}$ functions $\alpha_k, k \in \{1, \ldots, m\}$ such that*

$$\psi_m(\boldsymbol{x}, \theta) \geq 0,$$

*then the output layer of the neural ODE is invariant (i.e., $h(x_1(t)) \geq 0, \forall t \geq 0$) for all $\boldsymbol{x}(0)$ that satisfy $\psi_{k-1}(x_1, \ldots, x_k, \theta_{2,1}, \ldots, \theta_{k,k-1}) \geq 0, \forall k \in \{1, \ldots, m\}$.*

**Proof:** Given a safety constraint $b(x) \geq 0$, by Nagumo's Thm. (Nagumo, 1942), the necessary and sufficient condition for the system safety is $\dot{b}(x) \geq 0$, when $b(x) = 0$.

First, we have $\psi_m(\boldsymbol{x}, \theta) = \dot{\psi}_{m-1}(\boldsymbol{x}, \theta) + \alpha_m(\psi_{m-1}(\boldsymbol{x}, \theta)) \geq 0$ following Eqs. (5) and the constraint in Thm. 1. Since $\alpha_m$ is a class $\mathcal{K}$ function, $\alpha_m(\psi_{m-1}(\boldsymbol{x}, \theta))$ approaches 0 as $\psi_{m-1}(\boldsymbol{x}, \theta) \to 0$. In other words, we have $\dot{\psi}_{m-1}(\boldsymbol{x}, \theta) \geq 0$ when $\psi_{m-1}(\boldsymbol{x}, \theta) = 0$. Then, by Nagumo's Thm., we have that $\psi_{m-1}(\boldsymbol{x}, \theta) \geq 0$ is satisfied since the derivative of $\psi_{m-1}(\boldsymbol{x}, \theta)$ is non-decreasing on the hyperplane $\psi_{m-1}(\boldsymbol{x}, \theta) = 0$. Recursively, we show $\psi_k(\boldsymbol{x}, \theta) \geq 0, \forall k \in \{0, 1, \ldots, m-1\}$. Since $h(x_1) = \psi_0(\boldsymbol{x}, \theta)$, we have that $h(x_1(t)) \geq 0, \forall t \geq 0$. ■

Following the proof of Theorem 1, the invariances of hidden layers are also satisfied if the conditions in Theorem 1 are satisfied in addition to the output invariance. If $h(x_1(0)) > 0$, we can always find class $\mathcal{K}$ functions $\alpha_k, k \in \{1, \ldots, m\}$ such that $\psi_k(\boldsymbol{x}(0), \theta) \geq 0, \forall k \in \{1, \ldots, m\}$ (Xiao & Belta, 2022). These class $\mathcal{K}$ functions can be found recursively from $k = 0$ (i.e., $h(x_1)$) to $k = m$. In order to make sure that the remaining condition in Theorem 1 is satisfied for all $t \geq 0$, we can change the parameter $\theta$ by differentiable quadratic programs (QPs). This is shown in the invariance enforcing section. In this way, we map the output specification to a constraint on the parameter of neural ODEs.

**4.2 Invariance propagation for neural ODEs with external input I:** Note that we do not need to back-propagate the invariance from the output layer to the input layer. Instead, we can just back-propagate it to any hidden layer. For neural ODEs with input **I**, like (3), we can also back-propagate the invariance to the input **I** (suppose **I** is added to the input layer), as shown in the following theorem.

**Theorem** 2 *Given a neural ODE with input* ***I*** *defined by (3), the output specification $h(x_1) \geq 0$, and the functions $\psi_{k-1}$ defined by (5), if there exist class $\mathcal{K}$ functions $\alpha_k, k \in \{1, \ldots, m\}$, and any Lipschitz continuous input* ***I*** *satisfies*

$$\psi_m(\boldsymbol{x}, I, \theta) \geq 0,$$

*then the output layer of the neural ODE is invariant (i.e., $h(x_1(t)) \geq 0, \forall t \geq 0$) for all $\boldsymbol{x}(0)$ that satisfy $\psi_{k-1}(x_1, \dots, x_k, \theta_{2,1}, \dots, \theta_{k,k-1}) \geq 0, \forall k \in \{1, \dots, m\}$.*

**Proof:** The proof is identical to that of Theorem 1. ■

Theorem 2 propagates the invariance from the output layer to the external input **I** of a neural ODE through the constraint of Theorem 2. In order to make sure that $I$ is Lipschitz continuous, we can use differentiable QPs to filter it. This is shown in the following section.

**4.3 Enforcing invariances.** Enforcing the invariance of a neural ODE is equivalent to the satisfaction of the conditions in Theorems 1 and 2. Recall that the existence of all class $\mathcal{K}$ functions can be found recursively from the output layer to the input layer if $h(x_1(0)) > 0$. If $h(x_1(0)) \leq 0$, then the output of the neural ODE will be driven to satisfy $h(x_1) \geq 0$ when the constraint of Theorem 1 or the constraint of Theorem 2 is satisfied due to its Lyapunov property. The enforcing of the constraint of Theorem 1 or that of Theorem 2 could vary in different applications. We do not restrict to exact methods. We provide a minimum-deviation QP approach. We start with the case of neural ODEs without external input **I** and show how to enforce the constraint of Theorem 1 on parameters $\theta$.

**Enforcing invariance for neural ODEs without input I.** We aim to enforce the invariance on a subset of the parameters instead of all parameters as the computational complexity grows with $\mathcal{O}(n^3)$ in the number of parameters. Suppose we wish to enforce the invariance on parameters $\theta_{k,k-1}, k \in \{2, m+1\}$. Then, we only need to back-propagate the invariance to the layer $k-1$. The constraint in Theorem 1 then becomes $\psi_{k-1}(\boldsymbol{x}, \theta_{2,1}, \dots, \theta_{k,k-1}) \geq 0$, which is a nonlinear constraint on $\theta_{k,k-1}$ and is hard to solve. In order to formulate a differentiable QP that takes $\theta_{k,k-1}$ as the decision variable to enforce the invariance, we only drop the activation functions of layer $k-1$. Then, the neural ODE between layers $k$ and $k-1$ can be written in the form:

$$\dot{x}_{k-1} = \theta_{k,k-1} x_k + b_k \tag{6}$$

where $b_k$ is a bias. Thus, $\psi_{k-1}$ in (5) can be rewritten in the form:

$$\begin{aligned}\psi_{k-1}(\boldsymbol{x}, \theta_{2,1}, \dots, \theta_{k,k-1}) = \psi_{k-2,1}(\boldsymbol{x}, \theta_{2,1}, \dots, \theta_{k-1,k-2})\theta_{k,k-1} + \psi_{k-2,2}(\boldsymbol{x}, \theta_{2,1}, \dots, \theta_{k-1,k-2}) \\ + p_{k-1}\alpha_{k-1}(\psi_{k-2}(\boldsymbol{x}, \theta_{2,1}, \dots, \theta_{k-1,k-2})),\end{aligned} \tag{7}$$

where $\psi_{k-2,1}, \psi_{k-2,2}$ are the coefficient and drift terms for $\theta_{k,k-1}$, respectively, when taking the derivative of $\psi_{k-2}$ along the state space of the neural ODE (6) between layers $k$ and $k-1$. $p_{k-1} > 0$ is a trainable parameter that can make the differentiable QP not conservative (a case study for conservativeness is shown in experiments). The formulation is similar to a HOCBF that is defined over affine control dynamics, and we have similar affine neural ODEs in terms of $\theta_{k,k-1}$ if layer $k-1$ does not have an activation function (and only for layer $k-1$). Then, we can formulate the following optimization:

$$\min_{\theta_{k,k-1}} ||\theta_{k,k-1} - \theta^{\dagger}_{k,k-1}||^2 \tag{8}$$

$$\begin{aligned}\text{s.t. } \psi_{k-2,1}(\boldsymbol{x}, \theta_{2,1}, \dots, \theta_{k-1,k-2})\theta_{k,k-1} + \psi_{k-2,2}(\boldsymbol{x}, \theta_{2,1}, \dots, \theta_{k-1,k-2}) \\ + p_{k-1}\alpha_{k-1}(\psi_{k-2}(\boldsymbol{x}, \theta_{2,1}, \dots, \theta_{k-1,k-2})) \geq 0,\end{aligned} \tag{9}$$

where $||\cdot||$ denotes the Euclidean norm, $\theta^{\dagger}_{k,k-1}$ denotes the value of $\theta_{k,k-1}$ during training or after training. The above optimization becomes a QP with all other variables fixed except $\theta_{k,k-1}$ This solving method has shown to work in (Ames et al., 2014) (Glotfelter et al., 2017) (Xiao & Belta, 2022). At each discretization step, we solve the above QP and get $\theta^*_{k,k-1}$. Then we set $\theta_{k,k-1} = \theta^*_{k,k-1}$ during the testing of the neural ODE. This way, we can enforce the invariance, i.e., guarantee that $h(x_1(t)) \geq 0, \forall t \geq 0$.

Since the trainable parameter $p_{k-1}$ determines the conservativeness of the solving method, we would also like to tune it during the training of a neural ODE. As a result, we have a differentiable QP (Amos & Kolter, 2017). This training framework is shown in Figure 3.

**Enforcing invariance for neural ODEs with input I:** In this case, we try to enforce the condition in Thm. 2 for neural ODE (3). The constraint in Theorem 2 is also a nonlinear constraint on the input **I** if the input layer possesses an activation function. Thus, in order to apply a similar strategy as the above, we only drop the activation functions of the input layer. The neural ODE (3) can then be written as:

$$\dot{\boldsymbol{x}} = f_{NN_\theta}(\boldsymbol{x}) + g_{NN_\theta}(\boldsymbol{x})I, \tag{10}$$

Then, the constraint of Thm. 2 can be rewritten as the following formulation:

$$\psi_m(\boldsymbol{x}, I, \theta) = \psi_{m-1,1}(\boldsymbol{x}, \theta) I + \psi_{m-1,2}(\boldsymbol{x}, \theta) + p_m \alpha_m(\psi_{m-1}(\boldsymbol{x}, \theta)), \tag{11}$$

where $\psi_{m-1,1}, \psi_{m-1,2}$ are defined similarly as in (7). $p_m > 0$ is also a trainable parameter.

Finally, we can formulate the following optimization problem:

$$\min_{I} ||I - I^\dagger||^2 \tag{12}$$

$$\text{s.t. } \psi_{m-1,1}(\boldsymbol{x}, \theta) I + \psi_{m-1,2}(\boldsymbol{x}, \theta) + p_m \alpha_m(\psi_{m-1}(\boldsymbol{x}, \theta)) \geq 0, \tag{13}$$

where $I^\dagger$ denotes the original input of the neural ODE. The above optimization also becomes a differentiable QP if we fix all the other parameters except $I$. At each discretization step, we solve the QP and get $I^*$. Then, we set $I = I^*$ to replace the original input $I^\dagger$. This way, we enforce the invariance of neural ODEs with external input, i.e., guarantee that $h(x_1(t)) \geq 0, \forall t \geq 0$ after filtering the input $\mathbf{I}$ of the neural ODE by the above differentiable QP.

**4.4 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 parameter $p_{k-1}$ or $p_m$ in the differentiable QP (8) or (12), which is non-trivial when we have many output specifications for the neural ODE. 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 $\theta$, 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 $p_{k-1}$ in QP (8)) such that $\theta$ minimally deviates from $\theta^\dagger$. The parameter $p_{k-1}$ determines how large the deviation might be. The training of the network and the QP can be performed alternatingly. We summarize the training process in Figure 3.

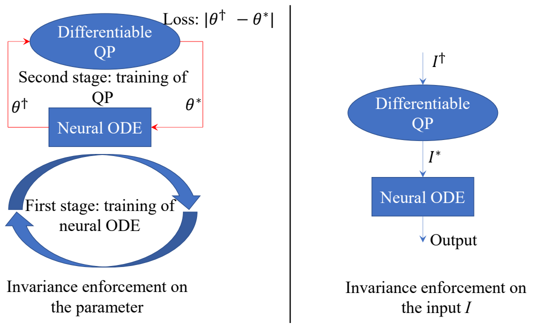

Differentiable QP
Loss: $|\theta^\dagger - \theta^*|$
Second stage: training of QP
$\theta^\dagger$
$\theta^*$
Neural ODE
First stage: training of neural ODE
Invariance enforcement on the parameter
$I^\dagger$
Differentiable QP
$I^*$
Neural ODE
Output
Invariance enforcement on the input $I$

Figure 3: Training neural ODEs with invariance. The training of a neural ODE with input $\mathbf{I}$ is regular when we enforce the invariance on $I$.

## 5 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:

$$\dot{\boldsymbol{y}} = f(\boldsymbol{y}, \boldsymbol{u}), \tag{14}$$

where $\boldsymbol{y} \in \mathbb{R}^q$ is the state of the system, $\boldsymbol{u} = x_1$ 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 (14) to satisfy the following (safety) constraint:

$$b(\boldsymbol{y}) \geq 0, \tag{15}$$

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 (15) for system (14). In other words, we map the safety constraint (15) onto the following HOCBF constraint:

$$\frac{d\phi_{d-1}}{d\boldsymbol{y}} f(\boldsymbol{y}, \boldsymbol{u}) + \alpha_d(\phi_{d-1}(\boldsymbol{y})) \geq 0, \tag{16}$$

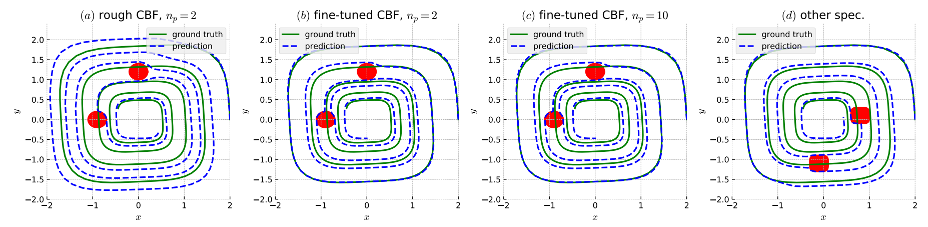


Figure 4: After-training invariance enforcing for spiral curve regression with output specifications. $n_p$ denotes the number of parameters randomly chosen in the QP (8). (a)-(c) are with two circular undesired sets defined by: $h_j(\boldsymbol{x}) = (x - x_{o_j})^2 + (y - y_{o_j})^2 - R^2$, $(x_{o_j}, y_{o_j})$ denotes the location of undesired set $j \in S, R > 0$, and (d) is with two randomly-placed superellipse-type undesired sets defined by: $h_j(\boldsymbol{x}) = (x - x_{o_j})^4 + (y - y_{o_j})^4 - R^4$.

Table 1: Spiral curve comparisons between neural ODE, neural ODE + shielding, and neural ODE + invariance. The focus is on the specification guarantees rather than accuracy of models.

| Item | In-loop training test error (↓) | Post-training test error(↓) | Deadlock rate (↓) | Complex spec. | Point-wise guarantee | Inter-sampling |
|---|---|---|---|---|---|---|
| Neural ODE (baseline)(Chen et al., 2018) | $0.489 \pm 0.165$ | $0.489 \pm 0.165$ | 0% | × | × | × |
| Shielding(Ferlez et al., 2020) | $0.611 \pm 0.120$ | $0.609 \pm 0.103$ | 5% | × | √ | × |
| Invariance (Ours) | $0.550 \pm 0.116$ | $0.443 \pm 0.082$ | 0% | √ | √ | √ |

where $\phi_k(\boldsymbol{y}) = \dot{\phi}_{k-1}(\boldsymbol{y}) + \alpha_k(\phi_{k-1}(\boldsymbol{y})), k = \{1, \dots, d-1\}$ and $\phi_0(\boldsymbol{y}) = b(\boldsymbol{y})$, $\alpha_k, k \in \{1, \dots, d\}$ are class $\mathcal{K}$ functions. It is worth noting that the construction of the HOCBF is similar to the construction (5) of the invariance of a neural ODE. The satisfaction of the above HOCBF constraint (16) implies the satisfaction of the safety constraint (15).

Since the output of the neural ODE is used to control the dynamical system, i.e., $\boldsymbol{u} = x_1$, we can find the output constraint of the neural ODE from (16) in the form:

$$h(x_1, \boldsymbol{y}) = \frac{d\phi_{d-1}}{d\boldsymbol{y}} f(\boldsymbol{y}, x_1) + \alpha_d(\phi_{d-1}(\boldsymbol{y})) \geq 0, \tag{17}$$

Then, we can back-propagate the invariance of the neural ODE (i.e., $h(x_1(t), \boldsymbol{y}(t)) \geq 0, \forall t \geq 0$) 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 (14) 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.

## 6 Experiments

In this section, we present four machine learning tasks to demonstrate the effectiveness of the proposed invariance of neural ODEs. We start with modeling spiral curve dynamics from data, where obstacles can be placed at random locations on the spiral's trajectories. We then show the invariance on two publicly available datasets generated by the Mujoco physics engine and show how a mathematical specification such as the convexity portrait can be enforced on the neural ODE using our invariance method. Finally, we demonstrate how to use our invariance method to ensure the safety of neural ODE controllers in autonomous driving tasks.

**6.1 Spiral Curve Regression with Specifications.** The training data comes from solving an ODE $[\dot{x}, \dot{y}]^T = A[x^3, y^3]^T$ as given in (Chen et al., 2018), where $A = [-0.1, -2.0; 2.0, -0.1]$. We use a neural ODE to fit the data. Suppose 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. This can be done after the training using the proposed invariance or during the 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 (8). As a result, the outputs of the neural ODE can satisfy all the specifications (see Figures 4 (a)-(d)). In the post-training test, we need to carefully choose the class $\mathcal{K}$ functions shown in (5). Otherwise, the resulting trajectory would be overly-conservative such that there is a large deviation from the original

Table 2: Walker2d-v2 and halfcheetah-v2 comparisons between neural ODE, neural ODE + truncation, and neural ODE + invariance. The focus is on the safety guarantees rather than accuracy of models.

| Item | Walker2d-v2 test error (↓) | Halfcheetah-v2 test error(↓) | Complex spec. | Point-wise guarantee | Inter-sampling |
|---|---|---|---|---|---|
| Neural ODE (baseline)(Chen et al., 2018) | $1.060 \pm 0.072$ | $2.171 \pm 0.026$ | $\times$ | $\times$ | $\times$ |
| Truncation Brockman et al. (2016) | $1.147 \pm 0.084$ | $2.170 \pm 0.026$ | $\times$ | $\surd$ | $\times$ |
| Invariance (Ours) | $1.057 \pm 0.071$ | $2.131 \pm 0.024$ | $\surd$ | $\surd$ | $\surd$ |

one even when the state is far away from the constraint boundary $h_j(\boldsymbol{x}) = 0, j \in S$, as shown in Fig. 4a. With slightly fined-tuned CBF parameters $p_i$ (i.e., its class $\mathcal{K}$ functions), the conservativeness can be addressed, as shown in Figure 4b.

We can also minimally change a different number of parameters of the neural ODE. Comparing Fig. 4b with Figure 4c, 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. In order to show the robustness of the proposed invariance, we also tested other types of specifications, such as the two randomly-placed superellipse-type undesired sets shown in Figure 4d. The outputs of the neural ODE can also guarantee the satisfaction of the corresponding output specifications. We also consider invariance in the training loop, which can address the conservativeness of the invariance by learning instead of non-trivial parameter tuning, as shown in Fig. 1 (also shown in Fig. 7 in Appendix).

The comparisons between our invariance, shielding (Ferlez et al., 2020), and the baseline neural ODE (Chen et al., 2018) are shown in Table 1 in which each of the results is evaluated using 100 trained models. Our invariance model can achieve better performance while avoiding deadlock (as shown in Fig. 7 in Appendix). Most importantly, the proposed invariance can guarantee output specifications, including addressing the inter-sampling effect, i.e., specification satisfaction between sampling time.

**6.2 HalfCheetah-v2 and Walker2d-v2 kinematic modeling.** In this section, we evaluate our invariance framework on two publicly available datasets for modeling physical dynamical systems Lechner & Hasani (2020); Hasani et al. (2021b). 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. 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 2, while guaranteeing the satisfaction of complex safety specifications. We enforced our invariance on 17, 34, and 170 model parameters, respectively, and the performance is similar.

**6.3 Convexity Portrait of a Function.** In this section, we demonstrate how we can make sure 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 following Jensen's inequality is satisfied:

$$\mu_1 g(x) + \mu_2 g(y) \geq g(\mu_1 x + \mu_2 y), \tag{18}$$

where $\mu_1 \in [0, 1], \mu_2 \in [0, 1]$ such that $\mu_1 + \mu_2 = 1$. We sample three values $(g(x), g(y), g(\mu_1 x + \mu_2 y))$ from the convex function $g(z) = z^2$, and use a neural ODE to fit these three values.

The in-distribution case study is shown in Appendix. Although the outputs of a trained neural ODE model may satisfy Jensen's inequality (18) 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 5c. However, with the proposed invariance, we can guarantee that the future prediction of the model also the satisfaction of Jensen's inequality (18) (see blue dashed line in Fig. 5c).

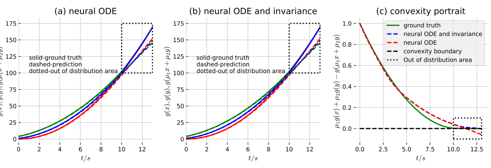
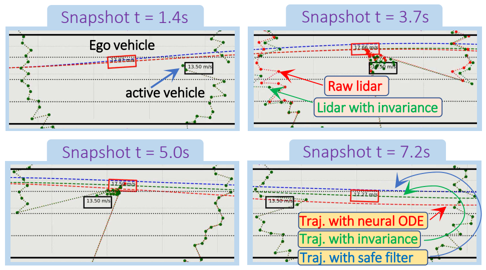
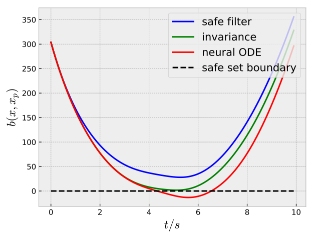


Figure 5: Convexity portrait of the neural ODE outputs. $x, y$ are functions of $t$. The non-negativity of the functions in (c) demonstrates the satisfaction of Jensen's inequality (18).



(a) Simulation snapshots.



(b) Safety portrait.

Figure 6: 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.

Table 3: Self-driving comparisons between safe filter, BarrierNet, neural ODE and invariance. The focus is on the safety guarantees and conservativeness rather than accuracy of models.

| item | Traj. test error (↓) | Conservativeness ($\geq 0$ & ↓) | Safety guarantees ($\geq 0$) | Dynamics free | Difficulty of training | Filtering approach | Non-conservative | Model complexity |
|---|---|---|---|---|---|---|---|---|
| Neural ODE(Chen et al., 2018) | $0.46 \pm 0.04$ | $-13.11 \pm 1.49$ | $-17.26$ | $\surd$ | low | $\times$ | $\surd$ | low |
| Safe filter (Pereira et al., 2020) | $0.96 \pm 0.04$ | $27.69 \pm 1.08$ | 24.60 | $\times$ | high | output | $\times$ | high |
| BarrierNet (Xiao et al., 2021) | $0.34 \pm 0.01$ | $8.51 \pm 0.33$ | 7.69 | $\times$ | high | output | $\surd$ | high |
| Invariance (Ours) | $0.36 \pm 0.01$ | $1.97 \pm 0.06$ | 1.83 | $\surd$ | medium | input | $\surd$ | low |

### 6.4 Lidar-based End-to-End Autonomous Driving.

In this section, we consider Lidar-based end-to-end autonomous driving. The neural ODE takes a Lidar point cloud as input **I**, and outputs controls for the autonomous vehicle to follow the lane. The states of the ego and edo 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. The problem and training setups are given in Appendix.

**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 6(a). The safety constraint $b(\boldsymbol{x}, \boldsymbol{x}_p)$ becomes negative (as the red curve shown in Fig. 6(b)) 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 **I**. The dimension of the Lidar information is $1 \times 100$, and thus, the dimension of the decision variable of the QP (12) 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 6(a). The modified Lidar information (through invariance) is illustrated by the green-dotted curve in the snapshot $t = 3.7s$ of Figure 6(a).

We also compare the proposed invariance with popular safe filter approaches (Pereira et al., 2020) (Xiao et al., 2021). 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 6(a)), and thus stay unnecessarily far away from the optimal trajectory (ground truth). The BarrierNet (Xiao et al., 2021) 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 3 that includes testing results of 100 case studies under noisy lidar perception. The invariance has the least conservativeness while guaranteeing safety.

**Limitations.** (a) The output specification of neural ODE may be unknown or hard to obtain, such as those from images, financial markets, etc. How to learn these output specifications is one of the remaining challenges to be further studied. (b) The proposed invariance is hard to be applied to non-continuous models, such as classical multi-layer perceptrons, convolutional neural networks, and discrete RNNs.

## REFERENCES

Victor M Martinez Alvarez, Rareş Roşca, and Cristian G Fălcuţescu. Dynode: Neural ordinary differential equations for dynamics modeling in continuous control. *arXiv preprint arXiv:2009.04278*, 2020.

A. D Ames, G. Notomista, Y. Wardi, and M. Egerstedt. Integral control barrier functions for dynamically defined control laws. *IEEE control systems letters*, 5(3):887–892, 2020.

Aaron D. Ames, Jessy W. Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs with application to adaptive cruise control. In *Proc. of 53rd IEEE Conference on Decision and Control*, pp. 6271–6278, 2014.

Alexander Amini, Igor Gilitschenski, Jacob Phillips, Julia Moseyko, Rohan Banerjee, Sertac Karaman, and Daniela Rus. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robotics and Automation Letters*, 5(2):1143–1150, 2020.

Brandon Amos and J. Zico Kolter. 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.

Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter. 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.

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

Jean-Pierre Aubin. *Viability theory*. Springer, 2009.

Justin Baker, Elena Cherkaev, Akil Narayan, and Bao Wang. Learning pod of complex dynamics using heavy-ball neural odes. *arXiv preprint arXiv:2202.12373*, 2022.

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

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 6572–6583, 2018.

Ricky TQ Chen, Brandon Amos, and Maximilian Nickel. Learning neural event functions for ordinary differential equations. *arXiv preprint arXiv:2011.03902*, 2020.

Xing Chen, Flavio Abreu Araujo, Mathieu Riou, Jacob Torrejon, Dafiné Ravelosona, Wang Kang, Weisheng Zhao, Julie Grollier, and Damien Querlioz. Forecasting the outcome of spintronic experiments with neural ordinary differential equations. *Nature communications*, 13(1):1–12, 2022.

Jyotirmoy V. Deshmukh, James P. Kapinski, Tomoya Yamaguchi, and Danil Prokhorov. 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.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. *Advances in Neural Information Processing Systems*, 32, 2019.

James Ferlez, Mahmoud Elnaggar, Yasser Shoukry, and Cody Fleming. Shieldnn: A provably safe nn filter for unsafe nn controllers. *preprint arXiv:2006.09564*, 2020.

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

Sophie Gruenbacher, Ramin Hasani, Mathias Lechner, Jacek Cyranka, Scott A Smolka, and Radu Grosu. On the verification of neural odes with stochastic guarantees. *arXiv preprint arXiv:2012.08863*, 2020.

Sophie Gruenbacher, Mathias Lechner, Ramin Hasani, Daniela Rus, Thomas A Henzinger, Scott Smolka, and Radu Grosu. Gotube: Scalable stochastic verification of continuous-depth models. *arXiv preprint arXiv:2107.08467*, 2021.

Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. *arXiv preprint arXiv:2006.04439*, 2020.

Ramin Hasani, Mathias Lechner, Alexander Amini, Lucas Liebenwein, Max Tschaikowski, Gerald Teschl, and Daniela Rus. Closed-form continuous-depth models. *arXiv preprint arXiv:2106.13898*, 2021a.

Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7657–7666, 2021b.

Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Neural certificates for safe control policies. *preprint arXiv:2006.08465*, 2020.

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

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

Suyong Kim, Weiqi Ji, Sili Deng, Yingbo Ma, and Christopher Rackauckas. Stiff neural ordinary differential equations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(9):093122, 2021.

Mathias Lechner and Ramin Hasani. Learning long-term dependencies in irregularly-sampled time series. *arXiv preprint arXiv:2006.04418*, 2020.

Mathias Lechner, Ramin Hasani, Manuel Zimmer, Thomas A Henzinger, and Radu Grosu. Designing worm-inspired neural networks for interpretable robotic control. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 87–94. IEEE, 2019.

Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas A Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2(10): 642–652, 2020.

Zhichao Li. Comparison between safety methods control barrier function vs. reachability analysis. *arXiv preprint arXiv:2106.13176*, 2021.

Lucas Liebenwein, Ramin Hasani, Alexander Amini, and Daniela Rus. Sparse flows: Pruning continuous-depth models. *Advances in Neural Information Processing Systems*, 34, 2021.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. *Advances in Neural Information Processing Systems*, 33:3952–3963, 2020.

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

Marcus Aloysius Pereira, Ziyi Wang, Ioannis Exarchos, and Evangelos A. Theodorou. Safe optimal control using stochastic barrier functions and deep forward-backward sdes. In *Conference on Robot Learning*, 2020.

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward fast continuous-depth models. *Advances in Neural Information Processing Systems*, 33: 21105–21117, 2020.

Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

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

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

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

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

Keng Peng Tee, Shuzhi Sam Ge, and Eng Hock Tay. Barrier lyapunov functions for the control of output-constrained nonlinear systems. *Automatica*, 45(4):918–927, 2009.

Emanuel Todorov, Tom Erez, and Yuval Tassa. 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.

Charles Vorbach, Ramin Hasani, Alexander Amini, Mathias Lechner, and Daniela Rus. Causal navigation by continuous-time neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

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

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

Wei Xiao and Calin Belta. 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.

Wei Xiao and Calin Belta. High-order control barrier functions. *IEEE Transactions on Automatic Control*, 67(7):3655–3662, 2022. doi: 10.1109/TAC.2021.3105491.

Wei Xiao, Ramin Hasani, Xiao Li, and Daniela Rus. Barriernet: A safety-guaranteed layer for neural networks. *preprint arXiv:2111.11277*, 2021.

Hengjun Zhao, Xia Zeng, Taolue Chen, Zhiming Liu, and Jim Woodcock. Learning safe neural network controllers with barrier certificates. *Form Asp Comp*, 33:437–455, 2021.

## A Appendix

### A.1 Spiral Curve Regression with Specifications

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:

$$\min_{a_1,a_2} (a_1 - A[0, 1])^2 + (a_2 - A[1, 0])^2 \tag{19}$$

s.t. CBF constraints:

$$\begin{aligned} 2(A[0, 1] - O_{x,1})A[1, 0]^3 a_1 + 2(A[1, 0] - O_{y,1})A[0, 1]^3 a_2 + h_1(x) \geq 0, \\ 2(A[0, 1] - O_{x,2})A[1, 0]^3 a_1 + 2(A[1, 0] - O_{y,2})A[0, 1]^3 a_2 + h_2(x) \geq 0, \end{aligned} \tag{20}$$

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.

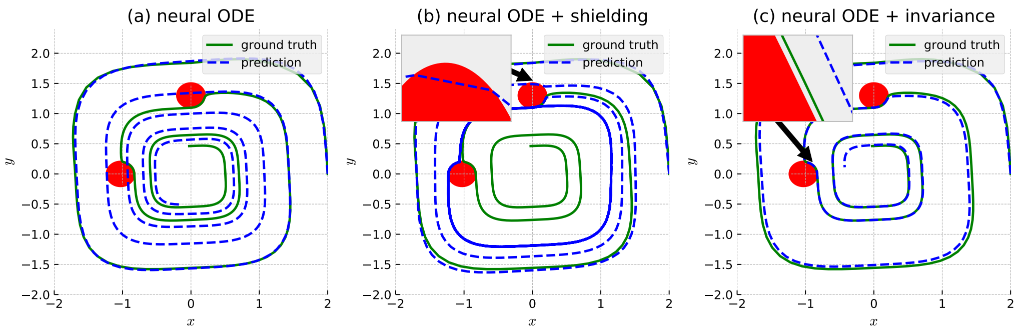


Figure 7: Invariance in training for spiral curve regression with output specifications. $h(\boldsymbol{x}) = \min\{h_j(\boldsymbol{x}), j \in S\} \geq 0$ denotes the satisfaction of output specifications. $h_j(\boldsymbol{x}) = (x - x_{o_j})^2 + (y - y_{o_j})^2 - R^2, j \in S$. The shielding method can only guarantee point-wise satisfaction, and thus the specification can still be violated between samplings. While the invariance can guarantee the satisfaction of specifications for all times. Deadlock may also happen in this case with the shielding method, in which case the trajectory repeatedly evolves around the obstacle.

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).

The training implementation and the enforcing QP for the invariance of the neural ODE are also given in the attached code. The $NN_\theta$ in the neural ODE is a three-layer fully connected network with sizes 2, 50, and 2, respectively. The activation functions used in the model are tanh.

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 in training.** The training data comes from solving the same ODE as above but also satisfies the two output specifications shown in Fig. 7. We use the proposed training pipeline shown in Figure 3 (left) to include the training of CBFs in the invariance. As expected, the model outputs from training a neural ODE cannot accurately approximate the ground truth and violate the output specifications (see Figure 7a). However, with invariance in the training loop, the model outputs can strictly satisfy the output specification while staying close to the ground truth (see Figure 7c). The shielding method cannot guarantee the satisfaction of specifications between sampling times.

## A.2 Convexity Portrait of a Function

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:

$$h(x, y) = \mu_1 z_1 + \mu_2 z_2 - z_3, \tag{21}$$

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. The $NN_\theta$ in the neural ODE is a three-layer fully connected network with sizes 3, 50, and 3, respectively. The activation functions used in the model are tanh.

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.

**In distribution.** Within the range of the training data, the trained neural ODE is not guaranteed to satisfy Jensen's inequality (18) as illustrated by the red-dashed curve in Figure 8c. However, with the proposed invariance, we can guarantee that the model outputs satisfy Jensen's inequality (through changing the parameter with the proposed QP approach (8)), as shown by the blue-dashed curve in Figure 8c.

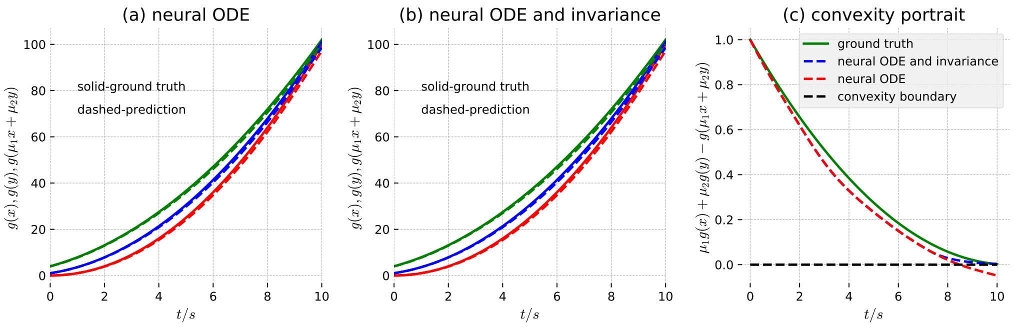


Figure 8: In-distribution convexity portrait of the neural ODE outputs. $x, y$ are functions of $t$. The data for $t \in [0, 10]$ is within the training set. The non-negativity of the functions in (c) demonstrates the satisfaction of Jensen's inequality (18).

## A.3 HalfCheetah-v2 and Walker2d-v2 kinematic modeling

We evaluate our invariance framework on two publicly available datasets for modeling physical dynamical systems Lechner & Hasani (2020); Hasani et al. (2021b). 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.

The $NN_\theta$ in the neural ODE is a three-layer fully connected network with sizes 17, 64, and 17, respectively. The activation functions used in the model are Tanh.

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.

## A.4 Lidar-based End-to-End Autonomous Driving

**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.** 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.

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 $2\pi$. 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.

**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.

The $NN_\theta$ in the neural ODE 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. 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.

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.