Reachability Analysis of a General Class of Neural Ordinary Differential Equations

Diego Manzanas Lopez[1], Patrick Musau[1], Nathaniel P. Hamilton[1], and Taylor T. Johnson[1]

Vanderbilt University, Nashville, TN 37212, USA
{diego.manzanas.lopez, taylor.johnson}@vanderbilt.edu

Abstract. Continuous deep learning models, referred to as Neural Ordinary Differential Equations (Neural ODEs), have received considerable attention over the last several years. Despite their burgeoning impact, there is a lack of formal analysis techniques for these systems. In this paper, we consider a general class of neural ODEs with varying architectures and layers, and introduce a novel reachability framework that allows for the formal analysis of their behavior. The methods developed for the reachability analysis of neural ODEs are implemented in a new tool called NNVODE. Specifically, our work extends an existing neural network verification tool to support neural ODEs. We demonstrate the capabilities and efficacy of our methods through the analysis of a set of benchmarks that include neural ODEs used for classification, and in control and dynamical systems, including an evaluation of the efficacy and capabilities of our approach with respect to existing software tools within the continuous-time systems reachability literature, when it is possible to do so.

1 Introduction

Neural Ordinary Differential Equations (ODEs) were first introduced in 2018, as a radical new neural network design that boasted better memory efficiency, and an ability to deal with irregularly sampled data [20]. The idea behind this family of deep learning models is that instead of specifying a discrete sequence of hidden layers, we instead parameterize the derivative of the hidden states using a neural network [8]. The output of the network can then be computed using a differential equation solver [8]. This work has spurred a whole range of follow-up work, and since 2018 several variants have been proposed, such as augmented neural ODEs (ANODEs) and their ensuing variants [37,11,15]. These variants provide a more expressive formalism by augmenting the state space of neural ODEs to allow for the flow of state trajectories to cross. This crossing, prohibited in the original framework, allows for the learning of more complex functions that were prohibited by the original neural ODE formulation [11].

Due to the potential that neural networks boast in revolutionizing the development of intelligent systems in numerous domains, the last several years have witnessed a significant amount of work towards the formal analysis of these models. The first set of approaches that were developed considered the formal verification of neural networks (NN), using a variety of techniques including reachability methods [43,44,3], and SAT techniques [28,29]. Thereafter, many researchers proposed novel formal method approaches for neural network control systems (NNCS), where the majority of methods utilized a combination of NN and hybrid system verification techniques [13,21,41,25,22,6]. Building on this work, a natural outgrowth is extending these approaches to analyze and verify neural ODEs, and some recent studies have considered the analysis of formal properties of neural ODEs. One such study aims to improve the understanding of the inner operation of these networks by analyzing and experimenting with multiple neural ODE architectures on different benchmarks [34]. Some studies have considered analyses of the robustness of neural ODEs such as [7] and [45], which evaluate the robustness of image classification neural ODEs and compare the efficacy of this class of network against other more traditional image classifier architectures. The first reachability technique targeted for neural ODEs presented a theoretical regime for verifying neural ODEs using Stochastic Lagrangian Reachability (SLR) [18]. This method is an abstraction-based technique that computes confidence intervals for the calculated reachable set with probabilistic guarantees. In a follow-up work, these methods were improved and implemented in a tool called Gotube [19], which is able to compute reach sets for longer time horizons than most state-of-the-art reachability tools. However, these methods only provide stochastic bounds on the reach sets, so there are no formal guarantees on the derived results.

To the best of our knowledge, this paper presents the first deterministic verification framework for a general class of neural ODEs with multiple continuous-time and discrete-time layers. In this work, we present our verification framework NNVODE that makes use of deterministic reachability approaches for the analysis of neural ODEs. Our methods are evaluated on a set of benchmarks with different architectures and conditions in the area of dynamical systems, control systems, and image classification. We also compare our results against three state-of-the-art verification tools when possible. In summary, the contributions of this paper are:

– We introduce a general class of neural ODEs that allows for the combination of multiple continuous-time and discrete-time layers.
– We develop NNVODE, an extension of NNV [43], to formally analyze a general class of neural ODEs using sound and deterministic reachable methods.
– We run an extensive evaluation on a collection of benchmarks within the context of time-series analysis, control systems, and image classification. We compare the results to Flow*, GoTube, and JuliaReach for neural ODE architectures where this is possible.

2 Background and Problem Formulation

Neural ODEs emerged as a continuous-depth variant of neural networks becoming a special case of ordinary differential equations (ODEs), where the derivatives are defined by a neural network, and can be expressed as:

$$\dot{z} = g(z), \qquad z(t_0) = z_0, \tag{1}$$

where the dynamics of the function $g : \mathbb{R}^j \to \mathbb{R}^p$ are represented as a neural network, and initial state $z_0 \in \mathbb{R}^j$, where j corresponds to the dimensionality of the states z. A *neural network* (NN) is defined to be a collection of consecutively connected $\mathcal{NN}$-Layers as described in Definition 1.

Definition 1 ($\mathcal{NN}$-Layer). *A $\mathcal{NN}$-**Layer** is a function h: $\mathbb{R}^j \to \mathbb{R}^p$, with input $x \in \mathbb{R}^j$, output $y \in \mathbb{R}^p$ defined as follows*

$$y = h(x) \tag{2}$$

where the function h is determined by parameters θ, typically defined as a tuple $\theta = \langle \sigma, \mathbf{W}, \mathbf{b} \rangle$ for fully-connected layers, where $\mathbf{W} \in \mathbb{R}^{j \times p}$, $\mathbf{b} \in \mathbb{R}^p$, and activation function σ: $\mathbb{R}^j \to \mathbb{R}^p$, thus the fully-connected $\mathcal{NN}$-Layer is described as

$$y = h(x) = \sigma(\mathbf{W}(x) + \mathbf{b}). \tag{3}$$

However, for other layers such as convolutional-type $\mathcal{NN}$-Layers, θ may include parameters like the filter size, padding, or dilation factor and the function h in (3) may not necessarily apply. For a formal definition and description of the reachability analysis for each of these layers that are integrated within NNVODE, we refer the reader to Section 4 of [40]. For the Neural ODEs (NODEs), we assume that g is Lipschitz continuous, which guarantees that the solution of $\dot{z} = g(z)$ exists. This assumption allows us to model g with m layers of continuously differentiable activation functions σ_k for $k \in \{1, 2, ..., m\}$ such as sigmoid, tanh, and exponential activation functions. Described in (1) is the notion of a NODE as introduced in [8] and an example is illustrated in Figure 1.

Definition 2 (NODE). *A **NODE** is a function $\dot{z} = g(z)$ with m fully-connected $\mathcal{NN}$-Layers, and it is defined as follows*

$$\dot{z} = g(z) = h_m(h_{m-1}(...h_1(z))), \tag{4}$$

where $g : \mathbb{R}^j \to \mathbb{R}^p$, $\sigma_k : \mathbb{R}^{j_k} \to \mathbb{R}^{p_k}$, $\boldsymbol{W}_k \in \mathbb{R}^{j_k \times p_k}$, and $\boldsymbol{b}_k \in \mathbb{R}^{p_k}$. For each layer k=$\{1, 2, \dots, m\}$, we describe the function h_k as in (3).

An example of NODE with m = 2 hidden layers, 2 inputs and 2 outputs is depicted in Figure 1.

2.1 General Neural ODE

The general class of neural ODEs (GNODEs) considered in this work is more complex than previously analyzed neural ODEs as it may be comprised of two types of layer: NODEs and $\mathcal{NN}$-Layers. We introduce a more general framework where multiple NODEs can make up part of the overall architecture along with other $\mathcal{NN}$-Layers, as described in Definition 3. This is the reachability problem subject of evaluation in this work.

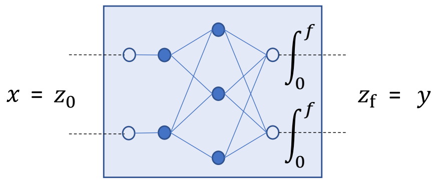


Fig. 1. Illustration of an example of NODE, as defined in (1) and (4), and Definition 2.

Definition 3 (GNODE). *A **GNODE** $\mathcal{F}$ is any sequence of consecutively connected N layers $\mathcal{L}_k$ for $k \in \{1, \ldots N\}$ with N_O NODEs and N_D $\mathcal{NN}$-Layers, that meets the conditions $1 \leq N_O \leq N$, $0 \leq N_D < N$, and $N_D + N_O = N$.*

With the above definition, we formulate a theorem for the restricted class of neural ODEs (NODE) that we use to compare our methods against existing techniques.

Observation 1 (Special case 1: NODE) *Let $\mathcal{F}$ be a GNODE with N layers. If $N = 1$, $N_O = 1$ and $N_D = 0$, then $\mathcal{F}$ is equivalent to an ODE whose continuous-time dynamics are defined as a neural network, and we refer to it as a **NODE** (as in Definition 2).*

In Figure 2, an example of a GNODE is shown, which has 1 input (x), 1 output (y), $N_O = 2$ NODEs, and $N_D = 5$ $\mathcal{NN}$-Layers, with its 5 numbered segments described as: 1) The first segment has a $\mathcal{NN}$-Layer, with one hidden layer of 2 neurons and an output layer of 2 neurons, 2) a NODE with one hidden layer of 3 neurons and an output layer of 2 neurons, 3) $\mathcal{NN}$-Layer with 3 hidden layers of 4,1, and 2 neurons respectively, and an output layer of 2 neurons, 4) NODE with a hidden layer of 3 neurons and output layer of 2 neurons, and 5) $\mathcal{NN}$-Layer with a hidden layer of 4 neurons and and output layer with 1 neurons.

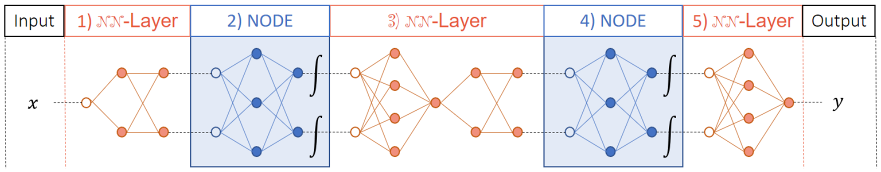


Fig. 2. Example of a GNODE. The filled circles represent weighted neurons in each layer, non-filled neurons represent the inputs of each layer-type segment, shown for visualization purposes.

Given this GNODE architecture, we are capable of encoding the originally proposed neural ODE [8], as well as several of its model improvements including higher-order neural ODEs like the second-order neural ODE (SONODE) [37], augmented neural ODEs (ANODEs) [11], and input-layer augmented neural ODEs (ILNODEs). This general class of neural ODEs from Definition 3 is applicable to many applications including image classification, time series prediction, dynamical system modeling and normalizing flows.

Observation 2 (Special case 2 - NNCS) *We can consider the NNCS as a special case of the* GNODE*, as the NNCS models also consist of $\mathcal{NN}$-Layers and NODE, under the assumption that the plant is described as an NODE. NNCS in the general architecture have $\mathcal{NN}$-Layers followed by an ODE or NODE, which connects back to the $\mathcal{NN}$-Layers in a feedback loop manner for cp control steps. Thus, by unrolling the NNCS cp times, we consecutively connect the $\mathcal{NN}$-Layers with the NODE, creating a* GNODE.

2.2 NODE: Applications

There are two application modes of a NODE, model reduction and dynamical systems. On one hand, we can use it as a substitute for multiple similar layers to reduce the depth and overall size of a model [8]. In this context, we treat the NODE as an input-output mapping function, and set the integration time t_f to a fixed number during training, which will be then used during simulation or reachability. Typically, this value is set to 1. On the other hand, we can make use of NODEs to capture the behavior of time-series data or dynamical systems. In this sense, t_f is not fixed, and it will be determined by the user/application. In summary, we can use NODEs for: 1) time-series or dynamical system modeling, and 2) model reduction. For time-series, t_f is variable, user or application dependent. For model reduction, t_f is a parameter of the model fixed before training. Only the output value at time $= t_f$ is used, while for the time-series models, we are interested in the interval $[0,t_f]$.

2.3 Reachability Analysis

The main focus of this manuscript is to introduce a framework for the reachability analysis of GNODEs. This framework combines set representations and methods from Neural Network (NN), Convolutional Neural Network (CNN) and hybrid systems reachability analysis. Similar to neural network reachability in NNV [43], we consider the set propagation through each layer as well as the set conversion between layers for all reachability methods for the supported layers. Hence, the reachability problem of GNODE is defined as follows:

Definition 4 (Reachable Set of a GNODE). *Let $\mathcal{F}\colon \mathbb{R}^j \to \mathbb{R}^p$ be a **GNODE** with N layers. The output reachable set $\mathcal{R}_N$ of a **GNODE** with input set $\mathcal{R}_0$ is defined as:*

$$\begin{aligned}
\mathcal{R}_1 &\triangleq \{y_1 \mid y_1 = f_1(y_0),\ y_0 \in \mathcal{R}_0\}, \\
\mathcal{R}_2 &\triangleq \{y_2 \mid y_2 = f_2(y_1),\ y_1 \in \mathcal{R}_1\}, \\
&\vdots \\
\mathcal{R}_N &\triangleq \{y_N \mid y_N = f_N(y_{N-1}),\ y_{N-1} \in \mathcal{R}_{N-1}\},
\end{aligned} \tag{5}$$

where f_i: $\mathbb{R}^{j_i} \rightarrow \mathbb{R}^{p_i}$ is the function f of layer i, where f is either a NODE (g) or a $\mathcal{NN}$-Layer (h).

The proposed solution to this problem is sound and incomplete, in other words, an over-approximation of the reachable set. This means that, given a set of inputs, we compute an output set of which, given any point in the input set, we simulate the GNODE and the output point is always contained within the reachable output set (sound). However, it is incomplete because the opposite is not true; there may be points in the output reachable set that cannot be traced back to be within the bounds of the input set.

Definition 5 (Soundness). *Let $\mathcal{F}$: $\mathbb{R}^j \rightarrow \mathbb{R}^p$ be a GNODE with an input set $\mathcal{R}_0$ and output reachable set $\mathcal{R}_f$. The computed $\mathcal{R}_f$ given $\mathcal{F}$ and $\mathcal{R}_0$ is **sound** iff $\forall x \in \mathcal{R}_0$, $\mid y = \mathcal{F}(x)$, $y \in \mathcal{R}_f$.*

Definition 6 (Completeness). *Let $\mathcal{F}$: $\mathbb{R}^j \rightarrow \mathbb{R}^p$ be a GNODE with an input set $\mathcal{R}_0$ and output reachable set $\mathcal{R}_f$. The computed $\mathcal{R}_f$ given $\mathcal{F}$ and $\mathcal{R}_0$ is **complete** iff $\forall x \in \mathcal{R}_0$, $\exists y = \mathcal{F}(x) \mid y \in \mathcal{R}_f$ and $\forall y \in \mathcal{R}_f$, $\exists x \in \mathcal{R}_0 \mid y = \mathcal{F}(x)$.*

Definition 7 (Reachable set of a $\mathcal{NN}$-Layer). *Let h: $\mathbb{R}^j \rightarrow \mathbb{R}^p$ be a $\mathcal{NN}-$ Layer as described in Definition 1. The reachable set $\mathcal{R}_h$, with input $\mathcal{X} \subset \mathbb{R}^n$ is defined as*

$$\mathcal{R}_h = \{y \mid y = h(x), x \in \mathcal{X}\}.$$

Definition 7 applies to any discrete-time layer that is part of a GNODE, including, but not limited to the following supported $\mathcal{NN} - Layers$ supported in NNVODE: fully-connected layers with ReLU, tanh, sigmoid and leaky-ReLU activation functions, and convolutional-type layers such as batch normalization, 2-D convolutional, and max-pooling.

The reachability analysis of NODEs is akin to the general reachability problem for any continuous-time system modeled by an ODE. If we represent the NODE as a single layer i of a GNODE with continuous dynamics described by (1), and assume that for a given initial state $z_0 \in \mathbb{R}^{j_i}$, the system admits a unique trajectory defined on $\mathbb{R}_0^+$, described by $\zeta(., z(t_0))$, then the reachable set of the given NODE can be characterized by Definition 8.

Definition 8 (Reachable set of a NODE). *Let g be a **NODE** with solution $\zeta(t; z_0)$ to (1) for initial state z_0. The reachable set, $\mathcal{R}_g$ at $t = t_F$, $\mathcal{R}_g(t_F)$, with initial set $\mathcal{R}_0 \subset \mathbb{R}^n$ at time $t = t_0$ is defined as*

$$\mathcal{R}_g(t_F) = \{\zeta(t; z_0) \in \mathbb{R}^n \mid z_0 \in \mathcal{R}_0, t \in [0, t_f]\}.$$

We also describe the reachable set for a time interval [t_0,t_f] as follows

$$\mathcal{R}_g([t_0, t_f]) := \bigcup_{t \in [t_0, t_f]} \mathcal{R}_g(t)$$

Now that we have outlined the general reachability problem of a NODE, whether we compute a single reachable set for the NODE at $t = t_F$, or over an interval $t \in [t_0,t_F]$ (computed by *get_time()*), depends on how the neural ODE was trained and the specific application of its use. However, the core computation remains the same. This is outlined in Algorithm 1.

Algorithm 1 Reachability analysis of a GNODE.

```
Input: $\mathcal{F}, \mathcal{R}_0$   // GNODE, input set
Output: $\mathcal{R}_f$   // output reachable set
 1: procedure $\mathcal{R}_f$ = REACH($\mathcal{F}, \mathcal{R}_0$)
 2:     N = $\mathcal{F}$.layers   // number of layers
 3:     for $i = 1 : N$ do   // loop through every layer
 4:         $\mathcal{L}_i = \mathcal{F}$.layer(i)
 5:         if $\mathcal{L}_i$ is NODE then   // check layer type
 6:             $\mathbf{t}_i$ = get_time($\mathcal{L}_i$)   // get integration time bounds of layer $i$
 7:             $\mathcal{R}_i = \text{reach}_{NODE}(\mathcal{L}_i, \mathcal{R}_{i-1}, \mathbf{t}_i)$   // reach set of layer $i$, Definition 8
 8:         else
 9:             $\mathcal{R}_i = \text{reach}_{\mathbb{NN}}(\mathcal{L}_i, \mathcal{R}_{i-1})$   // reach set of layer $i$, Definition 7
10:     $\mathcal{R}_f = \mathcal{R}_N$   // output reachable set
```

2.4 Reachability Methods

In the previous sections, we defined the reachable set of a GNODE using a layer-by-layer approach. However, the computation of these reachable sets is defined by the reachability methods and set representations utilized in their construction. For instance, the same operations are not utilized to compute the reachable set for a fully-connected layer with a hyperbolic tangent activation function as with a fully convolutional layer. In the following section, we describe the set of methods in the NNVODE tool available to the community to compute the reachable set of each specific layer.

We begin with the NODE approaches, where we make a distinction based on the underlying dynamics. If the NODE is nonlinear, we make use of zonotope and polynomial-zonotope based methods that are implemented and available in CORA [2,1]. If the NODE is purely linear, then we utilize the star set-based methods introduced in [5], which are more scalable than other zonotope-based methods and possess soundness guarantees as well.

For the $\mathbb{NN}$-Layers, there are several methods available, including zonotope-based and star-set based methods. However, we limited our implementation only using star sets to handle these layers, as this representation was demonstrated to be more computationally efficient and to enable tighter over-approximations of the derived reachable sets than zonotope methods [42]. Additionally, in using star set methods, we allow for the use of both approximate methods (*approx-star*) and exact methods (*exact-star*). A summary of these methods and supported layers is depicted in Table 1, and for a complete description of the reachability methods utilized in our work, we refer the reader to [43] and the manual[1][2].

Implementation. One of the key aspects of the verification of GNODEs is the proper encoding of the NODEs within reachability schemes. Depending

[1] NNV manual is available at: `https://github.com/verivital/nnv/blob/master/docs/manual.pdf`

[2] CORA manual (Release 2021) is available at: `https://tumcps.github.io/CORA/data/Cora2021Manual.pdf`

Table 1. Layers supported in NNVODE and reachability sets and methods available.

	Layer Type – **Set Rep. (method name)**
NODE	Linear – Star-set ("direct") [5]
NODE	Nonlinear – Zonotope, Polynomial Zonotope * [2,1]
$\mathcal{NN}$-Layer	FC: linear, ReLU – Star-set ("approx-star", "exact-star") [42]
$\mathcal{NN}$-Layer	FC: leakyReLU, tanh, sigmoid, satlin – Star-set ("approx-star") [43]
$\mathcal{NN}$-Layer	Conv2D – ImageStar ("approx-star", "exact-star") [43]
$\mathcal{NN}$-Layer	BatchNorm – ImageStar ("approx-star", "exact-star") [43]
$\mathcal{NN}$-Layer	MaxPooling2D – ImageStar ("approx-star", "exact-star") [43]
$\mathcal{NN}$-Layer	AvgPooling2D – ImageStar ("approx-star", "exact-star") [43]

* We support several methods available using Zonotope and PolyZonotopes, which includes user-defined fixed reachability parameters ("ZonoF", "PolyF") as well as adaptive reachability methods ("ZonoA" and "PolyA"), which require no prior knowledge on reach methods or systems to verify to produce relevant results.

on the software that is used, this process may vary and require distinct steps. As an example, some tools, like NNVODE, are simpler and allow for matrix multiplications within the definition of equations. However, for tools like Flow*, the set of steps required to properly encode this problem are more complex as it requires a definition for each individual equation of the state derivative. Thus, a more general conversion is needed, which is illustrated in the Appendix.

3 Evaluation

Having described the details of our reachability definitions, algorithm, and implementation, we now present the experimental evaluation of our proposed work. We begin by presenting, a method and tool comparison analysis against GoTube[3] [19], Flow*[4] [9] and JuliaReach[5] [6]. Then, we present a case study of an Adaptive Cruise Control system, and conclude with an evaluation of the scalability of our techniques using a random set of architectures for dynamical system applications as well as a set of classification models for MNIST. The GNODE architectures for each benchmark can be found in the Appendix. To facilitate the reproducibility of our experiments, we set a timeout of 2 hours (7200 seconds) for each reach set computation[6]. All our experiments were conducted on a desktop with the following configuration: Intel Core i7-7700 CPU @ 3.6GHz 8 core Processor, 64 GB Memory, and 64-bit Ubuntu 16.04.3 LTS OS.

3.1 Method and Tool Comparison

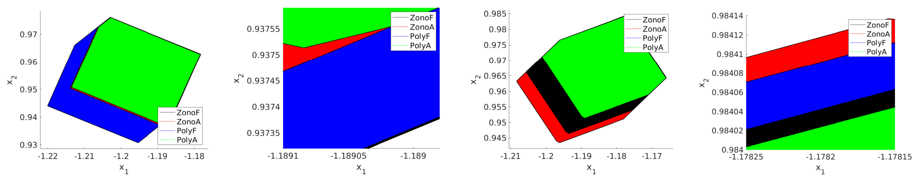
We have implemented several methods within NNVODE, and the first evaluation consists of comparing the available methods for nonlinear NODEs, fixed-step

[3] GoTube can be found at `https://github.com/DatenVorsprung/GoTube`
[4] Flowstar version 2.1.0 is available at `https://flowstar.org/`
[5] JuliaReach can be found at `https://juliareach.github.io/`
[6] Code to reproduce all results can be found here: `https://github.com/verivital/nnv/tree/master/code/nnv/examples/Submission/FORMATS2022`

zonotope and polynomial zonotopes (zono-F,poly-F) and adaptive zonotope and polynomial zonotope (zono-A,poly-A) based methods [2]. For all other $\mathcal{NN}$-Layers in the GNODEs, we use the star-set over-approximate methods. We considered multiple models, all inspired by the ILNODE representation that was introduced by Massaroli. They consist of a set of models with a varying number of augmented dimensions. All these models are instances of the GNODE class presented in Definition 3, which present an architecture of the form $\mathcal{NN}$-Layers + NODE + $\mathcal{NN}$-Layers. In this context, we were concerned with how the methods scale with respect to the number of dimensions of each model.

Table 2. Computation time of the reachability analysis of the Damped Oscillator benchmark. Results are shown in seconds with up to one decimal place.

Aug. Dims	Zono-F	Zono-A	Poly-F	Poly-A
0	**34.0**	574.5	201.7	654.0
1	**146.4**	4205.0	1573.9	3440.9
2	**441.0**	–	–	–



(a) Aug. Dims = 0 (b) Aug. Dims = 0 (c) Aug. Dims = 1 (d) Aug. Dims = 1

Fig. 3. Reach sets comparison of the Damped Oscillator benchmark of the model with 0 and 1 augmented dimensions. Plots (a) and (c) show the reachable set at $t = 1$s, and plots b) and d) show the zoomed-in reach sets to observe the minor size differences between the four methods: **ZonoA**, **ZonoF**, **PolyF** and **PolyA**.

In Table 2, we observe that the Zono-F method is the fastest across all models, while the adaptive methods are the slowest. Moreover, only Zono-F is able to complete the reach set computation for all three models, while the other methods time out. In terms of the size of the computed reach sets, we compare the last reach set obtained in Figure 3. In the subplots 3(b) and 3(d) we see the zoomed in reach sets, and observe that the Poly-A method computes the smallest over-approximate reach set across both experiments. Based on these results, in all subsequent experiments, we use the *zono-F* method for nonlinear NODEs, the *direct* method for linear NODEs, and the over-approximate star-set methods for all the $\mathcal{NN}$-Layers.

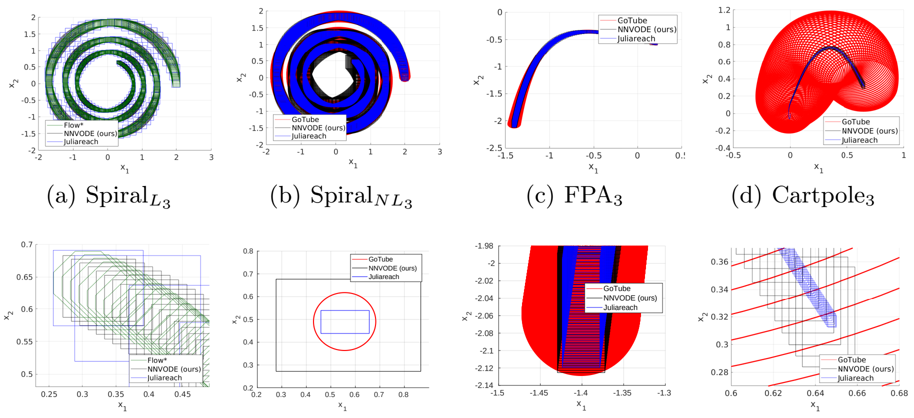
The next part of our evaluation consists of comparing NNVODE's methods for NODEs to those of Flow* [9], Gotube [19] and JuliaReach [6] across a collection of benchmarks that include a linear and nonlinear 2-dimensional spiral [8], a Fixed-Point Attractor (FPA) [36] and a controlled cartpole [17]. The computationn reachability results are displayed in Table 3 with the intention to characterize major differences between tools, i.e., to show some tools are 10× to 20× faster than others for some benchmarks. It is worth noting, that we are not experts on every tool that we considered. Thus it may be possible to optimize the reachable set computation for each benchmark with depending on the tool. However, we did not do so. Instead, we attempted around 3 to 5 different parameter combinations for each benchmark, and used the best results we could obtain when comparing against the other tools. The details can be found in the Appendix. The first three rows correspond to the linear spiral 2D model, and the subsequent three rows to the nonlinear one. The first set of observations that can be made in this context is that Flow* times out on all problems involving nonlinear neural ODEs, and that GoTube cannot obtain a solution to the reachability problem for the linear model. In terms of computation time, the results vary. Flow* is the fastest for the linear Spiral 2D model, regardless of the size of the initial set. In general, JuliaReach and NNVODE are much faster than GoTube, with JuliaReach being the fastest tool across the board. Additionally, there is not a significant difference between NNVODE and JuliaReach in the treatment of the nonlinear spiral model and FPA models. However, JuliaReach is an order of magnitude faster than NNVODE and two orders of magnitude faster than GoTube on the cartpole benchmark. In Figure 4 we displaya subset of the reachability results from Table 3 where we observe that JuliaReach is able to compute smaller over-approximations of the reachable set on all the benchmarks except for the linear spiral model. Notably, in most cases, GoTube computes the largest over-approximation, and this effect grows far more significantly than the other tools when the complexity of the model increases. This can be observed in Figures 4(d) and 4(h).

Table 3. Results of the reachability analysis of the NODE benchmarks. All results are shown in seconds. TH stands for Time Horizon and δ_μ to the input uncertainty.

Name	TH (s)	δ_μ	Flow*	GoTube	JuliaReach	NNVODE (ours)
Spiral$_{L_1}$	10	0.01	**4.0**	–	10.2	8.0
Spiral$_{L_2}$	10	0.05	**3.7**	–	7.2	7.4
Spiral$_{L_3}$	10	0.1	**3.7**	–	7.2	7.3
Spiral$_{NL_1}$	10	0.01	–	106.4	58.9	**47.4**
Spiral$_{NL_2}$	10	0.05	–	106.7	**41.7**	46.2
Spiral$_{NL_3}$	10	0.1	–	106.5	**41.9**	46.2
FPA$_1$	0.5	0.01	-	13.6	1.9	**1.3**
FPA$_2$	2.5	0.01	-	42.4	**5.6**	6.6
FPA$_3$	10.0	0.01	-	140.2	28.4	**7.5**
Cartpole$_1$	0.1	1e-4	-	183.0	**3.2**	67.9
Cartpole$_2$	1.0	1e-4	-	1590.9	**13.8**	404.3
Cartpole$_3$	2.0	1e-4	-	3065.7	**35.5**	834.7

3.2 Case Study: Adaptive Cruise Control (ACC)

This case study was selected to evaluate an original NNCS benchmark used in all the AINNCS ARCH-Competitions [26,27,33,43] against NNCS with NODEs as dynamical plants learned from simulation data using 3^{rd} order NODEs [37]. We demonstrate the verification of the ACC with different GNODEs learned as the plant model of the ACC and compare against the original benchmark, while using the same NN controller across all three models. The details of the original

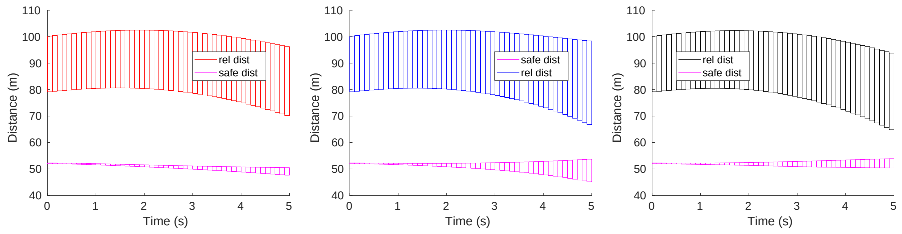


(a) Spiral$_{L3}$ (b) Spiral$_{NL3}$ (c) FPA$_3$ (d) Cartpole$_3$

(e) Spiral$_{L3}$ (f) Spiral$_{NL3}$ (g) FPA$_3$ (h) Cartpole$_3$

Fig. 4. NODE reach set comparisons. The top 4 figures ([a,d]) show the complete reachable sets of each benchmark, while the bottom 4 correspond to the zoomed-in reach sets (e, g) and to the zoomed-in figure of the last reach set (f,h). The figures show the computed reach sets of **GoTube**, **NNVODE**, **JuliaReach** and **Flow***.

ACC NNCS benchmark can be found in [43], and the architectures of the third order neural ODEs can be found in the Appendix.



(a) Original (b) Linear NODE (c) Nonlinear NODE

Fig. 5. Adaptive Cruise Control comparison. In red we display the relative distance of the original plant, in blue the linear NODE, in **black** the nonlinear NODE, and in magenta, the safe distance.

We considered all three models using the same initial conditions and present the results in Figure 5. The reachable sets obtained from all three plants are largely similar, and we can guarantee that all the models are safe since the intersection between the safe distance and the relative distance is empty. When we consider the size of the reachable sets, one can see that the original model returns the smallest reach set, whereas the linear model boasts the largest one. In all, the biggest difference between the plant dynamics is the computation time. Here, the linear 3^{rd} order NODE boasts the fastest computation time of 0.86

seconds, whereas the original plant takes 14.9 seconds, and the nonlinear 3^{rd} order NODE takes 998.7 seconds.

3.3 Classification NODEs

Our second set of experiments considered performing a robustness analysis for a set of MNIST classification models with only fully-connected $\mathcal{NN}$-Layers and other 3 models with convolutional layers as well. There is one linear NODE in each model, and we vary the number of parameters and states across all of them to study the scalability of our methods. We evaluate the robustness of these models under an L_∞ adversarial perturbation of $\epsilon = \{0.05, 1, 2\}$ over all the pixels, and $\epsilon = \{2.55, 12.75, 25.5\}$ over a subset of pixels (80). [7] The complete evaluation of this benchmark consists of a robustness analysis using 50 randomly sampled images for each attack. We compare the number of images the neural ODEs are robust to, as well as the total computation time to run the reachability analysis for each model in Table 4.

Definition 9 (Robustness). *Given a classification-based GNODE $\mathcal{F}(z)$, input $z \in \mathbb{R}^j$, perturbation parameter $\epsilon \in \mathbb{R}$ and an input set Z_p containing z_p such that $Z_p = \{z : ||z - z_p|| \leq \epsilon\}$ that represents the set of all possible perturbations of z. The neural ODE is locally* ***robust*** *at z if it classifies all the perturbed inputs z_p to the same label as z, i.e., the system is* ***robust*** *if $\mathcal{F}(z_p) = \mathcal{F}(z)$ for all $z_p \in Z_p$.*

Table 4. Robustness analysis of MNIST classification GNODEs under L_∞ adversarial perturbations. The accuracy and robustness results are described as percentage values between 0 and 1, and the time computation corresponds to the average time to compute the reachable set per image. Columns 3-8 corresponds to $\epsilon = \{0.5,1,2\}$ over all pixels in the image, columns 9-14 corresponds to $\epsilon = \{2.55,12.75,25.5\}$ attack over a subset of pixels (80) n each image.

		ℓ_∞						$\ell_{\infty\ (80)}$					
		0.5		**1**		**2**		**2.55**		**12.75**		**25.5**	
Name	**Acc.**	**Rob.**	**T(s)**	**Rob.**	**T(s)**	**Rob.**	**T(s)**	**Rob.**	**T(s)**	**Rob.**	**T(s)**	**Rob.**	**T(s)**
FNODE$_S$	0.9695	1	0.0836	0.98	0.1062	0.98	0.1186	1	0.0192	0.98	0.0193	0.98	0.0295
FNODE$_M$	0.9772	1	0.0948	1	0.1177	0.98	0.2111	1	0.0189	1	0.0225	0.98	0.0396
FNODE$_L$	0.9757	1	0.1078	1	0.1674	0.96	0.5049	1	0.0187	1	0.0314	0.96	0.0709
CNODE$_S$	0.9706	0.98	13.95	0.96	15.82	0.86	17.58	0.98	1.943	0.94	3.121	0.78	4.459
CNODE$_M$	0.9811	1	176.5	1	193.8	1	293.8	1	21.45	1	83.28	1	182.0
CNODE$_L$	0.9602	1	1064	1	1089	1	1736	1	234.6	1	522.7	1	779.3

Finally, we performed a scalability study using a set of random GNODE architectures with multiple NODEs. Here, we focus on the nonlinear methods for both the $\mathcal{NN}$-Layers and the NODEs and evaluate how our methods scale with the number of neurons, inputs, outputs and dimensions in the NODE. The main challenge of these benchmarks is the presence of multiple NODEs within the GNODE. There are a total of 6 GNODEs (XS,S,M,L,XL, and XXL), all with the same number of layers. However, we increase the number of inputs, outputs, and parameters across all the layers of the GNODEs. Here, XS corresponds to

[7] Adversarial perturbations are applied before normalization, pixel values $z_p \in [0,255]$.

the smallest model and XXL to the largest. A description of these architectures can be found in the Appendix.

Several trends can be observed from Table 5. The first is that in general, smaller models have smaller reach set computation times, with two notable exceptions: the first run with XS and the last experiment with XL. Furthermore, one can observe that the largest difference in the reach set computation times comes from increasing the number of states in the NODEs, which are $\{2, 3, 4, 4, 5, 5\}$ for both NODEs in every model in {XS,S,M,L,XL, and XXL} respectively, while increasing the input and output dimensions ($\{1, 2, 2, 3, 3, 4\}$ respectively) does not affect the reachability computation as much. This is because of the complexity of nonlinear ODE reachability as state dimensions increase, while for $\mathcal{NN}-Layers$, increasing the size of the inputs or neurons by 1 or a few units does not affect the reachability computation as much.

Table 5. Computation time of the reachability analysis of the randomly generated GNODEs. Results are shown in seconds.

	XS	**S**	**M**	**L**	**XL**	**XXL**
$\delta_\mu = 0.01$	57.0	16.2	59.3	61.8	168.3	223.9
$\delta_\mu = 0.02$	3.4	11.4	42.2	41.0	262.4	115.4
$\delta_\mu = 0.04$	3.1	10.3	37.6	72.9	1226.3	243.6

4 Related Work

Analysis of Neural ODEs. To the best of our knowledge, this is the first empirical study of the formal verification of neural ODEs as presented in the general neural ODE (GNODE) class. Some other works have analyzed neural ODEs, but are limited to a more restricted class of neural ODEs with only purely continuous-time models. We refer to these models in this paper as NODEs. The most comparable work is a theoretical inquiry of the neural ODE verification problem using Stochastic Lagrangian Reachability (SLR) [18], which was later extended and implemented in a stochastic reachability analysis tool called GoTube [19]. The SLR method is an abstraction-based technique that is able to compute a tight over-approximation of the set of reachable states, and provide stochastic guarantees in the form of confidence intervals for the derived reachable set. Beyond reachability analysis, there have also been several works investigating the robustness of neural ODEs. In [7], the robustness of neural ODE image classifiers is empirically evaluated against residual networks, which are a standard deep learning model for image classification tasks. Their analysis demonstrates that neural ODEs are more robust to input perturbations than residual networks. In a similar work, a robustness comparison of neural ODEs and standard CNNs was performed [45]. Their work considers two standard adversarial attacks, Gaussian noise and FGSM [16], and their analysis illustrate that neural ODEs are more robust to adversarial perturbations. In terms of the analysis of GNODEs, to the best of our knowledge, no comparable work as been done, although for specific models where all the $\mathcal{NN}$-Layers of a GNODE have fully-connected layers with continuous differentiable activation functions like sigmoid or tanh, it may be possible to compare our methods to other tools (with minor modifications) like Verisig [25,22] or JuliaReach [6]. However, that would restrict the more general class of neural ODEs (GNODEs) that we evaluate in this manuscript.

Table 6. Summary of related verification tools. A ✓means that the tool supports verification of this class, a ◯ means that it may be supported it, but some minor changes may be needed, a – means it does not support it, and a ⊙ means that some small changes have been made to the tool for comparison and the tools has been used to verify at least one example on this class.

Tool	**ODE**[8]	**NODE**[9]	**NN**[10]	**NNCS**	**GNODE**
CORA [2]	✓	⊙	–	–	–
ERAN [39]	–	–	✓	–	–
Flow* [9]	✓	⊙	–	–	–
GoTube [19]	✓	✓	–	–	–
JuliaReach [6]	✓	⊙	✓	✓	–
Marabou [29]	–	–	✓	–	–
nnenum [3]	–	–	✓	–	–
ReachNN [21,13]	✓	◯	✓	✓	–
Reluplex [28]	–	–	✓	–	–
ReluVal [44]	–	–	✓	–	–
Sherlock [12]	✓	◯	✓	✓	–
SpaceEx [14]	✓	◯	–	–	–
Verisig [25,22]	✓	◯	✓	✓	–
NNV [43]	✓	⊙	✓	✓	–
NNVODE (ours)	✓	✓	✓	✓	✓

[8]ODE verification is considered to be supported for any tool than can verify at least one of linear or nonlinear continuous-time ODEs. [9]NODE is a specific type of ODE, so in theory any tool that supports ODE verification may be able to support NODE. However, these tools are optimized for NODE and in practice may not be able to verify most of these models as seen on our comparison with Flow*. [10]We include in this category any tool that supports one or more verification methods for fully-connected, convolutional or pooling layers.

Verification of Neural Networks and Dynamical systems. The considered GNODEs are a combination of dynamical systems equations (ODEs) modeled using neural networks, and neural networks. When these subjects are treated in isolation, one finds that there are numerous studies that consider the verification of dynamical systems, and correspondingly there are numerous works that deal with the neural network verification problem. With respect to the former, the hybrid systems community has considered the verification of dynamical systems for decades and developed tools such as SpaceEx [14], Flow* [9], CORA [2], and JuliaReach [6] that deal with the reachability problem for discrete-time, continuous-time or even hybrid dynamics. A more comprehensive list of tools can be found in the following paper [10]. Within the realm of neural networks, the last several years have witnessed numerous promising verification methods proposed towards reasoning about the correctness of their behavior. Some representative tools include Reluplex [28], Marabou [29], ReluVal [44], NNV [43] and ERAN [39]. The tools have drawn inspiration from a wide range of techniques including optimization, search, and reachability analysis [32]. A discussion of these approaches, along with pedagogical implementations of existing methods, can be found in the following paper [32]. Building on the advancements of these fields, as a natural progression, frameworks that consider the reachability problem involving neural networks and dynamical systems have also emerged. Problems within the space are typically referred to as Neural Network Control Systems (NNCS), and some representative tools include NNV [43], Verisig [25,22], and ReachNN* [21,13]. These tools have demonstrated their capabilities and efficiency in several works including the verification of an adaptive cruise control (ACC) [43], an automated emergency breaking system [41], and autonomous racing cars [23,24] among others, as well as participated in the yearly challenges of NNCS verification [33,27,26]. These studies and their respective frameworks are very closely related to the work contained herein. In a sense, they deal with a restricted GNODE architecture, since their analysis combines the basic operations of neural network and dynamical system reachability in a feedback-loop manner. However, this restricted architecture would only consist of a fully-connected neural network followed by a NODE. In Table 6, we present a **comparison to verification tools** that can support one or more of the following verification problems: the analysis of continuous-time models with linear, nonlinear or hybrid dynamics (ODE), neural networks (NN), neural network control systems (NNCS), neural ODEs (NODE) and GNODEs as presented in Figure 2.

5 Conclusion & Future Work.

We have presented a verification framework to compute the reachable sets of a general class of neural ODEs (GNODE) that no other existing methods are able to solve. We have demonstrated through a comprehensive set of experiments the capabilities of our methods on dynamical systems, control systems and classification tasks, as well as the comparison to state-of-the-art reachability analysis tools for continuous-time dynamical systems (NODE). One of the main challenges we faced was the scalability of the nonlinear ODE reachability analysis as the dimension complexity of the models increased, as observed in the cartpole and damped oscillator examples. Possible improvements include integrating other methods into our framework such as [31], which improves the current nonlinear reachability analysis via an improved hybridization technique that reduces the sizes of the linearization domains, and therefore reduces overapproximation error. Another approach would be to make use of the Koopman Operator linearization prior to analysis in order to compute the reach sets of the linear system, which are easier and faster to compute as observed from the experiments conducted using the two-dimensional spiral benchmark [4]. In terms of models and architectures, latent neural ODEs [38] and some of its proposed variations such as controlled neural DEs [30] and neural rough DEs [35] have demonstrated great success in the area of time-series prediction, improving the performance of NODEs, ANODEs and other deep learning models such as RNNs or LSTMs in time-series tasks. The main idea behind these models is to learn a *latent* space from the input of the neural ODE from which to sample and predict future values. In the future, we will analyze these models in detail and explore the addition of verification techniques that can formally analyze their behavior.

References

1. Althoff, M. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control* (New York, NY, USA, 2013), HSCC '13, Association for Computing Machinery, p. 173–182.
2. Althoff, M. An introduction to cora 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems* (2015).
3. Bak, S. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium* (2021), Springer, pp. 19–36.
4. Bak, S., Bogomolov, S., Duggirala, P. S., Gerlach, A. R., and Potomkin, K. Reachability of black-box nonlinear systems after koopman operator linearization. In *7th IFAC Conference on Analysis and Design of Hybrid Systems, ADHS 2021, Brussels, Belgium, July 7-9, 2021* (2021), R. M. Jungers, N. Ozay, and A. Abate, Eds., vol. 54 of *IFAC-PapersOnLine*, Elsevier, pp. 253–258.
5. Bak, S., and Duggirala, P. S. Simulation-equivalent reachability of large linear systems with inputs. In *Computer Aided Verification* (Cham, 2017), R. Majumdar and V. Kunčak, Eds., Springer International Publishing, pp. 401–420.
6. Bogomolov, S., Forets, M., Frehse, G., Potomkin, K., and Schilling, C. Juliareach: a toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control* (2019), pp. 39–44.
7. Carrara, F., Caldelli, R., Falchi, F., and Amato, G. On the robustness to adversarial examples of neural ode image classifiers. In *2019 IEEE International Workshop on Information Forensics and Security (WIFS)* (2019), pp. 1–6.
8. Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. *Advances in Neural Information Processing Systems* (2018).
9. Chen, X., Ábrahám, E., and Sankaranarayanan, S. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification* (Berlin, Heidelberg, 2013), N. Sharygina and H. Veith, Eds., Springer Berlin Heidelberg, pp. 258–263.
10. Doyen, L., Frehse, G., Pappas, G. J., and Platzer, A. Verification of hybrid systems. In *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Springer, 2018, pp. 1047–1110.
11. Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural odes. In *Advances in Neural Information Processing Systems* (2019), H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc.
12. Dutta, S., Chen, X., and Sankaranarayanan, S. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control* (New York, NY, USA, 2019), HSCC '19, ACM, pp. 157–168.
13. Fan, J., Huang, C., Chen, X., Li, W., and Zhu, Q. Reachnn*: A tool for reachability analysis of neural-network controlled systems. In *International Symposium on Automated Technology for Verification and Analysis* (2020), Springer, pp. 537–542.
14. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., and Maler, O. Spaceex: Scalable verification of hybrid systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV)* (2011), S. Q. Ganesh Gopalakrishnan, Ed., LNCS, Springer.

15. Gholaminejad, A., Keutzer, K., and Biros, G. Anode: Unconditionally accurate memory-efficient gradients for neural odes. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19* (7 2019), International Joint Conferences on Artificial Intelligence Organization, pp. 730–736.
16. Goodfellow, I., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations* (2015).
17. Gruenbacher, S., Cyranka, J., Lechner, M., Islam, M. A., Smolka, S. A., and Grosu, R. Lagrangian reachtubes: The next generation, 2020.
18. Gruenbacher, S., Hasani, R. M., Lechner, M., Cyranka, J., Smolka, S. A., and Grosu, R. On the verification of neural odes with stochastic guarantees. In *AAAI* (2021).
19. Gruenbacher, S., Lechner, M., Hasani, R., Rus, D., Henzinger, T. A., Smolka, S., and Grosu, R. Gotube: Scalable stochastic verification of continuous-depth models, 2021.
20. Hao, K. A radical new neural network design could overcome big challenges in ai, Apr 2020.
21. Huang, C., Fan, J., Li, W., Chen, X., and Zhu, Q. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Trans. Embed. Comput. Syst. 18*, 5s (Oct. 2019).
22. Ivanov, R., Carpenter, T., Weimer, J., Alur, R., Pappas, G., and Lee, I. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *Computer Aided Verification* (Cham, 2021), A. Silva and K. R. M. Leino, Eds., Springer International Publishing, pp. 249–262.
23. Ivanov, R., Carpenter, T. J., Weimer, J., Alur, R., Pappas, G. J., and Lee, I. *Case Study: Verifying the Safety of an Autonomous Racing Car with a Neural Network Controller.* Association for Computing Machinery, New York, NY, USA, 2020.
24. Ivanov, R., Jothimurugan, K., Hsu, S., Vaidya, S., Alur, R., and Bastani, O. Compositional learning and verification of neural network controllers. *ACM Trans. Embed. Comput. Syst. 20*, 5s (sep 2021).
25. Ivanov, R., Weimer, J., Alur, R., Pappas, G. J., and Lee, I. Verisig: Verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control* (New York, NY, USA, 2019), HSCC '19, ACM, pp. 169–178.
26. Johnson, T. T., Lopez, D. M., Benet, L., Forets, M., Guadalupe, S., Schilling, C., Ivanov, R., Carpenter, T. J., Weimer, J., and Lee, I. Arch-comp21 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)* (2021), G. Frehse and M. Althoff, Eds., vol. 80 of *EPiC Series in Computing*, EasyChair, pp. 90–119.
27. Johnson, T. T., Lopez, D. M., Musau, P., Tran, H.-D., Botoeva, E., Leofante, F., Maleki, A., Sidrane, C., Fan, J., and Huang, C. Arch-comp20 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)* (2020), G. Frehse and M. Althoff, Eds., vol. 74 of *EPiC Series in Computing*, EasyChair, pp. 107–139.
28. Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification* (2017), Springer, pp. 97–117.

29. Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D. L., Kochenderfer, M. J., and Barrett, C. The marabou framework for verification and analysis of deep neural networks. In *Computer Aided Verification* (Cham, 2019), I. Dillig and S. Tasiran, Eds., Springer International Publishing, pp. 443–452.
30. Kidger, P., Morrill, J., Foster, J., and Lyons, T. Neural Controlled Differential Equations for Irregular Time Series. *Advances in Neural Information Processing Systems* (2020).
31. Li, D., Bak, S., and Bogomolov, S. Reachability analysis of nonlinear systems using hybridization and dynamics scaling. In *International Conference on Formal Modeling and Analysis of Timed Systems* (2020), FORMATS '20, Springer.
32. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., and Kochenderfer, M. J. Algorithms for verifying deep neural networks. *Foundations and Trends in Optimization 4*, 3-4 (2021), 244–404.
33. Lopez, D. M., Musau, P., Tran, H.-D., Dutta, S., Carpenter, T. J., Ivanov, R., and Johnson, T. T. Arch-comp19 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems* (April 2019), G. Frehse and M. Althoff, Eds., vol. 61 of *EPiC Series in Computing*, EasyChair, pp. 103–119.
34. Massaroli, S., Poli, M., Park, J., Yamashita, A., and Asama, H. Dissecting neural odes. In *Advances in Neural Information Processing Systems* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., pp. 3952–3963.
35. Morrill, J., Salvi, C., Kidger, P., Foster, J., and Lyons, T. Neural rough differential equations for long time series, 2021.
36. Musau, P., and Johnson, T. T. Continuous-time recurrent neural networks (ctrnns) (benchmark proposal). In *5th Applied Verification for Continuous and Hybrid Systems Workshop (ARCH)* (Oxford, UK, july 2018).
37. Norcliffe, A., Bodnar, C., Day, B., Simidjievski, N., and Lió, P. On second order behaviour in augmented neural odes. In *Advances in Neural Information Processing Systems* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., pp. 5911–5921.
38. Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. K. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems* (2019), H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc.
39. Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. Fast and effective robustness certification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2018), NIPS'18, Curran Associates Inc., p. 10825–10836.
40. Tran, H.-D., Bak, S., Xiang, W., and Johnson, T. T. Verification of deep convolutional neural networks using imagestars. In *32nd International Conference on Computer-Aided Verification (CAV)* (July 2020), Springer.
41. Tran, H.-D., Cei, F., Lopez, D. M., Johnson, T. T., and Koutsoukos, X. Safety verification of cyber-physical systems with reinforcement learning control. In *ACM SIGBED International Conference on Embedded Software (EMSOFT'19)* (October 2019), ACM.
42. Tran, H.-D., Musau, P., Lopez, D. M., Yang, X., Nguyen, L. V., Xiang, W., and Johnson, T. T. Star-based reachability analysis for deep neural networks.

In *23rd International Symposium on Formal Methods (FM'19)* (October 2019), Springer International Publishing.

43. Tran, H.-D., Yang, X., Lopez, D. M., Musau, P., Nguyen, L. V., Xiang, W., Bak, S., and Johnson, T. T. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *32nd International Conference on Computer-Aided Verification (CAV)* (July 2020).
44. Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)* (2018), pp. 1599–1614.
45. Yan, H., Du, J., Tan, V. Y. F., and Feng, J. On robustness of neural ordinary differential equations, 2020.

A Appendix

A.1 Implementation Example of a NODE

Let the function $g(z)$ be composed of two fully-connected layers, where σ_1 and σ_2 are *tanh* and *linear* activation functions, respectively. The state, $z \in \mathbb{R}^2$, is a two-dimensional vector, the first layer contains five neurons and the second layer has two neurons. Therefore, the parameters of the layers are $W_1 \in \mathbb{R}^{5\times 2}$, $\mathbf{b}_1 \in \mathbb{R}^5$, $W_2 \in \mathbb{R}^{2\times 5}$ and $\mathbf{b}_2 \in \mathbb{R}^2$. We can define the state derivatives of this NODE as:

$$\begin{aligned} \dot{z} = g(z) = L_2(L_1(z)) &= \sigma_2(W_2 \times (\sigma_1(W_1 \times z + \mathbf{b}_1) + \mathbf{b}_2) \\ &= W_2 \times (tanh(W_1 \times z + \mathbf{b}_1) + \mathbf{b}_2. \end{aligned} \tag{6}$$

For NNVODE as well as JuliaReach or GoTube, the representation in (6) is a valid format. However, for consistency and comparison purposes, other tools like Flow* require the explicit equation for every state z_i. Let the states z, weights W_k and biases $\mathbf{b}_k$ be

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, W_1 = \begin{bmatrix} w_{1_{11}} & w_{1_{12}} \\ w_{1_{21}} & w_{1_{22}} \\ w_{1_{31}} & w_{1_{32}} \\ w_{1_{41}} & w_{1_{42}} \\ w_{1_{51}} & w_{1_{52}} \end{bmatrix}, \mathbf{b}_1 = \begin{bmatrix} b_{1_1} \\ b_{1_2} \\ b_{1_3} \\ b_{1_4} \\ b_{1_5} \end{bmatrix},$$

$$W_2 = \begin{bmatrix} w_{2_{11}} & w_{2_{12}} & w_{2_{13}} & w_{1_{14}} & w_{2_{15}} \\ w_{2_{21}} & w_{2_{22}} & w_{2_{23}} & w_{1_{24}} & w_{2_{25}} \end{bmatrix}, \mathbf{b}_2 = \begin{bmatrix} b_{2_1} \\ b_{2_2} \end{bmatrix},$$

then we can explicitly write out the full equations for every state derivative:

$$\begin{gathered} \dot{z}_1 = w_{2_{11}} tanh(w_{1_{11}} z_1 + w_{1_{12}} z_2 + b_{1_1}) + w_{2_{12}} tanh(w_{1_{21}} z_1 \\ + w_{1_{22}} z_2 + b_{1_2}) + w_{2_{13}} tanh(w_{1_{31}} z_1 + w_{1_{32}} z_2 + b_{1_3}) \\ + w_{1_{14}} tanh(w_{1_{41}} z_1 + w_{1_{42}} z_2 + b_{1_4}) \\ + w_{2_{15}} tanh(w_{1_{51}} z_1 + w_{1_{52}} z_2 + b_{1_5}) + b_{2_1}, \\ \dot{z}_2 = w_{2_{21}} tanh(w_{1_{11}} z_1 + w_{1_{12}} z_2 + b_{1_1}) + w_{2_{22}} tanh(w_{1_{21}} z_1 \\ + w_{1_{22}} z_2 + b_{1_2}) + w_{2_{23}} tanh(w_{1_{31}} z_1 + w_{1_{32}} z_2 + b_{1_3}) \\ + w_{1_{24}} tanh(w_{1_{41}} z_1 + w_{1_{42}} z_2 + b_{1_4}) \\ + w_{2_{25}} tanh(w_{1_{51}} z_1 + w_{1_{52}} z_2 + b_{1_5}) + b_{2_1}, \end{gathered}$$

For the special case where σ_k is linear for every layer $k \in m$, we can simplify the NODE as a linear ODE:

$$\begin{aligned}
\dot{z} =& (W_m W_{m-1} \dots W_1)\, z + \mathbf{b}_m + \mathbf{b}_{m-1} W_m \\
&+ \mathbf{b}_{m-2} W_m W_{m-1} + \mathbf{b}_1 W_m \dots W_2 \\
\dot{z} =& (W_m W_{m-1} \dots W_1)\, z + \mathbf{b}_1 W_m \dots W_2 \\
&+ \mathbf{b}_2 W_m \dots W_3 + \mathbf{b}_{m-1} W_m + \mathbf{b}_m \\
\dot{z} =& \prod_{i=1}^{m} (W_i) z + \sum_{i=1}^{m-1} (\mathbf{b}_i \prod_{j=i+1}^{m} (W_j)) + \mathbf{b}_m, \\
\dot{z} =& Az + Bu + c,
\end{aligned} \tag{7}$$

where A is equal to weights term, $A = \prod_{i=1}^{m} (W_i)$, $B = 0$, and the rest of the terms of the equation corresponds to the constant vector
$c = \sum_{i=1}^{m-1} \left(\mathbf{b}_i \prod_{j=i+1}^{m} (W_j)\right) + \mathbf{b}_m$.

A.2 Neural ODE architectures

In this section we describe the architectures used for all the results explained in Section 3. We utilized the same base structure to explain all GNODEs, which consist on a set of $\mathcal{NN}$-Layers, then a NODE, and then another set of $\mathcal{NN}$-Layers, with the exception of the random experiments, which have a total of two NODEs. The names of the layers are abbreviated to include the full architecture in the tables, where layer (ns_i) corresponds to a weighted layer with activation function *layer* and ns_i neurons. If there is no number next to the layer it means that only the activation function is applied, no weights corresponding to that layer. When a set of layers are inside brackets, it means that this subset of layers is consecutively repeated T times. For the description and architectures of the models of the fixed point attractor (FPA) and cartpole, we refer the reader to [36] and [17] respectively.

Table 7. GNODE architectures of the spiral 2D benchmark, all the models of the damped oscillator (D.Osc.$_{n_{aug}}$), where $n_{aug} \in \{0,1,2\}$ corresponds to the number of augmented dimensions, and 6 classification models trained on MNIST.

	$\mathcal{NN}$-Layers	**NODE**	**$\mathcal{NN}$-Layers**
Spiral$_L$	N/A	fc (10) - fc (2)	N/A
Spiral$_N L$	N/A	tanh (10) - fc (2)	N/A
D.Osc.$_{n_{aug}}$	fc (2+n$_{aug}$)	fc (20) - fc (20) -fc (2+n$_{aug}$)	fc(2)
FNODE$_S$	relu (64) - relu (10)	fc (10)	softmax (10)
FNODE$_M$	relu (64) - relu (32) - fc (16)	fc (10) - fc (16)	softmax (10)
FNODE$_L$	relu (64) - [relu (32)] [x3] - fc (16)	[fc (10)] [x3] - fc (16)	softmax (10)
CNODE$_S$	[conv2d (4,3) - BN - relu] [x2] - flatten	fc (10) - fc (676)	softmax (10)
CNODE$_M$	[conv2d (10,3) - BN - relu] [x2] - flatten	fc (10) - fc (1690)	softmax (10)
CNODE$_L$	[conv2d (16,3) - BN - relu] [x2] - flatten	fc (10) - fc (2704)	softmax (10)

ACC. The ACC GNODEs used as plant models for the NNCS reachability are represented as a 3rd order NODE, building upon prior knowledge of the ego and lead car

Table 8. GNODE architectures of the randomly generated GNODE with multiple NODEs. Next to the model, in parenthesis, is the input size of the GNODE.

	$\mathcal{NN}$-**Layer**	**NODE**	$\mathcal{NN}$-**Layer**	**NODE**	$\mathcal{NN}$-**Layer**
XS (1)	tanh(2)	tanh(2) - tanh(2)	tanh(2)	tanh(2)	tanh(1)
S (2)	tanh(3)	tanh(5) - tanh(3)	tanh(3)	tanh(3)	tanh(2)
M (2)	tanh(4)	tanh(8) - tanh(4)	tanh(4)	tanh(4)	tanh(2)
L (3)	tanh(4)	tanh(8) - tanh(4)	tanh(4)	tanh(4)	tanh(3)
XL (3)	tanh(5)	tanh(10) - tanh(5)	tanh(5)	tanh(5)	tanh(3)
XXL (4)	tanh(5)	tanh(10) - tanh(5)	tanh(5)	tanh(5)	tanh(4)

dynamics. The dynamics are defined in (8), and the NODE architectures are described in Table 9.

$$\begin{aligned}
\dot{x}_1 &= \dot{x}_{\text{lead}} = v_{\text{lead}}, \\
\dot{x}_2 &= \dot{v}_{\text{lead}} = \gamma_{\text{lead}}, \\
\dot{x}_3 &= \dot{\gamma}_{\text{lead}} = \dot{y}_1, \\
\dot{x}_4 &= \dot{x}_{\text{ego}} = v_{\text{ego}}, \\
\dot{x}_5 &= \dot{v}_{\text{ego}} = \gamma_{\text{ego}}, \\
\dot{x}_6 &= \dot{\gamma}_{\text{ego}} = \dot{y}_2, \\
\dot{y} &= g_{acc}(z_{acc}), \\
z_{acc} &= [\gamma_{\text{lead}}, \gamma_{\text{ego}}, a_{\text{lead}}, a_{\text{ego}}]^\intercal
\end{aligned} \tag{8}$$

where $acc = \{\text{L, NL}\}$, corresponding to the linear and nonlinear NODEs.

Table 9. Architectures of the ACC 3^{rd} order NODEs, where g_L represents the NODE architecture of the linear model and g_{NL} the nonlinear one.

	NODE
g_{NL}	tanh(10) - tanh(4)
g_L	fc(20) - fc(4)

A.3 Evaluation - Reachability Analysis Parameters

In this section we describe the parameters used for each of the tools we compare against, as well as the the parameters used in NNVODE. For each benchmark, we create an ordered list of parameters based on the order as these are presented in the corresponding results table. If there is only one value for the parameters, it means that all the reachability parameters are kept constant throughout the benchmark experiments.

Spiral2D

Flow*

- Reach step: $\{0.01, 0.01, 0.01, 0.002, 0.002, 0.002\}$
- Taylor Models: $\{8, 8, 8, [6, 20], [6, 20], [6, 20], \}$

GoTube

- Reach step: 0.01
- Batch size: 1000
- gamma: 0.01
- mu: 1.5

JuliaReach

- alg: TMJets21a
- abstol: $1e^{-10}$
- orderT: 5
- orderQ: 1
- Max steps: 3500
- No parameters were specified for the linear model (LinearTS), we run all instances with the default parameters using $solve(problem, t_F)$

Fixed Point Attractor (FPA)

Flow*

- Reach step: 0.01
- Taylor Models: $\{8, 20, 30\}$

GoTube

- Reach step: 0.01
- Batch size: 1000
- gamma: 0.01
- mu: 1.5

JuliaReach

- alg: TMJets21a
- abstol: $1e^{-10}$
- orderT: 5
- orderQ: 1
- Max steps: $\{400, 1700, 3500\}$

Cartpole

Flow*

- Time step: 0.01
- Taylor models: $\{30, 20, 8\}$

GoTube

- Reach step: 0.01
- Batch size: 1000

- gamma: 0.01
- mu: 1.5

JuliaReach

- alg: TMJets21a
- abstol: $1e^{-10}$
- orderT: 5
- orderQ: 1
- Max steps: $\{400, 1700, 3500\}$