# POLAR-Express: Efficient and Precise Formal Reachability Analysis of Neural-Network Controlled Systems

Yixuan Wang*, Weichao Zhou*, Jiameng Fan, Zhilu Wang, Jiajun Li,
Xin Chen, Chao Huang, Wenchao Li, Qi Zhu

***Abstract*—Neural networks (NNs) playing the role of controllers have demonstrated impressive empirical performance on challenging control problems. However, the potential adoption of NN controllers in real-life applications has been significantly impeded by the growing concerns over the safety of these neural-network controlled systems (NNCSs). In this work, we present POLAR-Express, an efficient and precise formal reachability analysis tool for verifying the safety of NNCSs. POLAR-Express uses Taylor model arithmetic to propagate Taylor models (TMs) layer-by-layer across a neural network to compute an over-approximation of the neural network. It can be applied to analyze any feed-forward neural networks with continuous activation functions, such as ReLU, Sigmoid, and Tanh activation functions that cover the common benchmarks for NNCS reachability analysis. Compared with its earlier prototype POLAR, we develop a novel approach in POLAR-Express to propagate TMs more efficiently and precisely across ReLU activation functions, and provide parallel computation support for TM propagation, thus significantly improving the efficiency and scalability. Across the comparison with six other state-of-the-art tools on a diverse set of common benchmarks, POLAR-Express achieves the best verification efficiency and tightness in the reachable set analysis. POLAR-Express is publicly available at https://github.com/ChaoHuang2018/POLAR_Tool.**

***Index Terms*—Neural-Network Controlled Systems; Reachability Analysis; Safety Verification; Formal Methods**

## I. Introduction

Neural networks (NNs) have been successfully used for decision making in a variety of systems such as autonomous vehicles [1], [2], [3], aircraft collision avoidance systems [4], robotics [5], HVAC control [6], [7], and other autonomous cyber-physical systems (CPSs) [8], [9]. NN controllers can be obtained using machine learning techniques such as reinforcement learning [10], [11], imitation learning [12], [13], and transfer learning [14]. However, the usage of NN controllers raises new challenges in verifying the safety of these systems due to the nonlinear and highly parameterized nature of neural networks and their closed-loop formations with dynamical systems [15], [16], [17], [18], and adversarial perturbations [19], [20], [21]. In this work, we consider the reachability verification problem of neural-network controlled systems (NNCSs).

Uncertainties around the state, such as those inherent in state measurement or localization systems, or scenarios where the system can start from anywhere in an initial space, require the consideration of an *initial state set* rather than a single initial state for the reachability problem. Specifically, we define the reachability problem for NNCSs as follows.

**Definition 1** (Reachability Problem of NNCSs)**.** *The reachability problem of an NNCS is to determine whether the system can reach a given goal state set from any state within an initial state set of the system, whereas the bounded-time version of this problem is to determine the reachability within a given bounded-time horizon.*

We show Fig 1 as an example of this set-based closed-loop reachability analysis. It is worth noting that simulation-based testings [22], which sample initial states from the initial state set, cannot provide formal safety guarantees such as "no system trajectory from the initial state set will lead to an obstacle collision." In this paper, we consider reachability analysis as the class of techniques that aim at tightly over-approximating the set of all reachable states of the system starting from an initial state set.

Reachability analysis of general NNCSs is notoriously hard due to nonlinearities that exist in both the NN controller and the physical plant. The closed-loop coupling of the NN controller with the plant adds another layer of complexity. To obtain a tight over-approximation of the reachable sets, reachability analysis needs to *track state dependencies across the closed-loop system and across multiple time steps*. While this problem has been well studied in traditional closed-loop systems without NN controllers [23], [24], [25], [26], [27], [28], it is less clear whether it is important to track the state dependency in NNCSs and how to track the dependency efficiently given the complexity of neural networks. This paper aims to bring clarity to these questions by comparing different approaches for solving the NNCS reachability problems.

Yixuan Wang* and Weichao Zhou* contributed equally to this work. Yixuan Wang, Zhilu Wang, and Qi Zhu are with the Department of Electrical and Computer Engineering at Northwestern University, Evanston, IL, USA (email: {yixuanwang2024@u., zhilu.wang@u, qzhu@}northwestern.edu). Weichao Zhou, Jiameng Fan, and Wenchao Li are with the Department of Electrical and Computer Engineering at Boston University, Boston, MA, USA (email: {zwc662, jmfan, wenchao}@bu.edu). Jiajun Li and Chao Huang are with the Department of Computer Science at the University of Liverpool, Liverpool, UK (email: {chao.huang2, j.li234}@liverpool.ac.uk). Xin Chen is with the Department of Computer Science, University of New Mexico, Albuquerque, NM, USA (email: {chenxin@unm.edu}).
We gratefully acknowledge the support from the National Science Foundation awards CCF-1646497, CCF-1834324, CNS-1834701, CNS-1839511, IIS-1724341, CNS-2038853, ONR grant N00014-19-1-2496, the US Air Force Research Laboratory (AFRL) under contract number FA8650-16-C-2642. This work is also supported by the grant EP/Y002644/1 under the EPSRC ECR International Collaboration Grants program, funded by the International Science Partnerships Fund (ISPF) and the UK Research and Innovation.

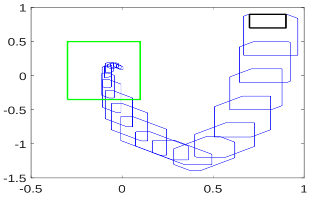



Fig. 1: An illustrating example of the reachability problem. The system shown in the figure can start from any state in the initial state set (black). Each polygon (blue) is the over-approximation of the state set after a control step by the NN controller. In this example, the over-approximation of the reachable set falls in the target set (green). Therefore, this goal-set-reaching verification problem is proven to be True.

Existing reachability analysis techniques for NNCSs typically use reachability analysis methods for dynamical systems as subroutines. For general nonlinear dynamical systems, the problem of *exact reachability is undecidable* [29]. Thus, methods for reachability analysis of nonlinear dynamical systems aim at computing a *tight over-approximation of the reachable sets* [30], [31], [32], [33], [34], [25], [27], [26]. On the other hand, there is also rich literature on verifying neural networks. Most of these verification techniques boil down to the problem of estimating or over-approximating the *output ranges* of the network [35], [36], [37], [38], [39], [40]. The existence of these two bodies of work gives rise to a straightforward combination of NN output range analysis with reachability analysis of dynamical systems for solving the NNCS reachability problem. However, early works have shown that this naive combination with a non-symbolic interval-arithmetic-based [41] output range analysis suffers from large over-approximation errors when computing the reachable sets of the closed-loop system [16], [15]. The primary reason is the lack of consideration of the interactions between the NN controller and the plant dynamics. Recent advances in the field of NN verification feature more sophisticated techniques that can yield tighter output range bounds and track the input-output dependency of an NN via symbolic bound propagation [38], [42], [40]. This opens up the possibility of improvement for the aforementioned combination strategy by substituting the non-symbolic interval-arithmetic-based technique with these new symbolic bound estimation techniques.

New techniques have also been developed to directly address the verification challenge of NNCSs. Early works mainly use *direct end-to-end over-approximation* [16], [15], [43] of the neural-network function, i.e. computing a function approximation of the neural network with guaranteed error bounds. While this approach can better capture the input-output dependency of a neural network compared to output ranges, it suffers from efficiency and scalability problems due to the need to sample from the input space. This approach is superseded by more recent techniques that leverage *layer-by-layer propagation* in the neural network [17], [44], [45], [46]. Layer-by-layer propagation techniques have the advantage of being able to exploit the structure of the neural network. They are primarily based on propagating *Taylor models* (TMs) layer by layer via Taylor model arithmetic to more efficiently obtain a function over-approximation of the neural network.

**Scope and Contributions:** We present POLAR-Express, a significantly enhanced version of our earlier prototype POLAR [46]. Inherited from POLAR [46], POLAR-Express uses layer-by-layer propagation of TMs to compute function over-approximations of NN controllers. Our technique is applicable to general feed-forward neural networks with continuous (but not necessarily differentiable) activation functions. Compared with POLAR, POLAR-Express has the following new features.

- A more efficient and precise method for propagating TMs across non-differentiable ReLU activation functions.
- Multi-threading support to parallelize the computation in the layer-by-layer propagation of Taylor models for neural-network controllers, which significantly improved the efficiency and scalability of our approach for complex systems.
- Comprehensive experimental evaluation with recent state-of-the-art tools such as RINO [47], CORA [48], and Juliareach [49]. Across a diverse set of benchmarks and tools, POLAR-Express achieves state-of-the-art verification efficiency and tightness of over-approximation in the reachable set analysis, outperforming all existing tools.

More specifically, compared with the existing literature [40], [18], [48], [49], [47], [46], [44], we provide the most comprehensive experimental evaluation across a wide variety of NNCS benchmarks including NN controllers with different activation functions and dynamical systems with up to 12 states. In terms of the over-approximation approach for NN, existing tools can be categorized into two classes. The first class shares the common idea of integrating NN output range analysis techniques with reachability analysis tools for dynamical systems, such as $\alpha,\beta$-CROWN [40], NNV [18], CORA [48], JuliaReach [49], and RINO [47]. The second class focuses on passing symbolic dependencies across the NNCS and across multiple control steps during reachability analysis, such as POLAR-Express and Verisig 2.0 [44]. Through the comparisons, we hope this paper can also serve as an accessible introduction to these analysis techniques, for those who wish to apply them to verify NNCSs in their own applications and for those who wish to dive more deeply into the theory of NNCS verification.

## II. Background

We first introduce the technical preliminaries and review existing techniques for the safety verification of NNCSs. An NNCS is often defined by an ODE that is governed by a feed-forward neural network at discrete times. Although it is undecidable to know if a state is reachable for an NNCS starting from an initial state, we may compute an over-approximated set of reachable states. The safety of an NNCS can be proven by showing that the reachable set of this NNCS does not contain any unsafe state. Although more general safety or robustness of an NNCS can be proven by computing an invariant for the system reachable states [50], [51], [52], [53], it is still hard to handle a large number of system variables and general nonlinear dynamics. Hence, most

**Algorithm 1:** Reachable set computation for NNCSs based on set propagation.

**Input:** Definition of the system modules, number of control steps $K$, the initial state set $X_0$.
**Output:** Over-approximation of the reachable set in $K$ steps.

$\mathcal{X}_0 \leftarrow X_0$, $\mathcal{R} \leftarrow \emptyset$; # the resulting over-approximate set
**for** $i = 0$ to $K - 1$ **do**
    Computing an over-approximation $\mathcal{U}_i$ for the NN output w.r.t. the input set $\mathcal{X}_i$;
    Computing a set $\mathcal{F}$ of flowpipes for the plant dynamics from the initial set $\mathcal{X}_i$ in a control step;
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{F}$;
    Evaluating an over-approximation for the reachable set after the control step and assigning it to $\mathcal{X}_{i+1}$;
**end for**
**return** $\mathcal{R}$.

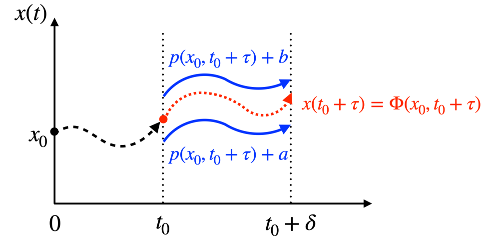



Fig. 2: Taylor model over-approximation of a flowmap.

of the existing methods for reachability analysis use the *set propagation* scheme [54]. That is, an over-approximation of the reachable set in a bounded time horizon can be obtained by iteratively computing a super-set for the reachable set in a time step and propagating it to the set computation for the next step. More precisely, starting from a given initial set $X_0$, a set propagation method computes an over-approximation of the reachable set $\Phi_{t\in[0,\delta]}(X_0, t)$ where $\delta$ is the time step, $\Phi$ denotes the system's evolution function (flowmap) that is often unknown. It then repeats the above work from the obtained reachable set over-approximation at the end of the previous step and computes a new over-approximation for the current one. The over-approximation segments are called *flowpipes*. Such a scheme has been proven effective in handling various system dynamics and efficient in handling large numbers of state variables [16], [18], [45], [49], [46], [47].

Algorithm 1 shows the main framework of set propagation for NNCSs. Starting with a given initial set $X_0$, the main algorithm repeatedly performs the following two main steps to compute the flowpipes in the $(i + 1)$-th control step for $i = 0, 1, \ldots, K-1$: (a) *Computing the range $\mathcal{U}_i$ of the control input.* This task is to compute the output range of the NN controller w.r.t. the current system state. Since the current system state is a subset of the latest flowpipe, $\mathcal{U}_i$ is computed as an over-approximate set. (b) *Flowpipe construction for the continuous dynamics.* According to the obtained range $\mathcal{U}_i$ of the constant control inputs, the reachable set in the current control step can be obtained using a flowpipe computation method for ODEs.

Existing methods can be mainly classified into the following two groups based on their over-approximation purposes.

**(I) Pure range over-approximations for reachable sets.** The techniques in this group aim at directly over-approximating the range of the reachable set using geometric or algebraic representations such as intervals [55], zonotopes [56] or other sets represented by constraints. Such an approach can often be developed by designing the over-approximation methods for the plant and controller individually and then using a higher-level algorithm to make the two methods work cooperatively for the closed-loop system. Many existing tools for computing the reachable set over approximations under the continuous dynamics defined by linear or nonlinear ODEs can be used to handle the plant, such as VNODE-LP[24], SpaceEx [25], CORA [26], and Flow* [27]. On the other hand, the task of computing the output range of a neural network can be handled by various output range analysis techniques developed in recent years [57], [58], [59], [60], [35], [36], [61], [62], [39], [38], [18], [42], [40]. The main advantages of the techniques in this group are twofold. First, there is no need to develop a new technique from scratch, and the correctness of the composed approach can be proven easily based on the correctness of the existing methods for the subtasks. Second, the performance of the approach is often good on simple case studies since it can use well-engineered tools as primitives. However, since those methods mainly focus on the pure range over-approximation work, and do not just lightly track the dependencies among the state variables under the system dynamics, it may accumulate significant over-approximation error when the plant dynamics is nonlinear or the initial set is large, making the resulting bounds less useful in proving properties of interest.

**(II) Functional over-approximations for system evolution.** The reachable set over-approximation methods in this category focus on a more challenging task than only over-approximating the reachable set ranges. They seek to compute an over-approximate function for the flowmap $\Phi$ of an NNCS. As we pointed out in the previous section, $\Phi$ is a function only in the variables representing the initial state and the time, and it often does not have a closed-form expression. However, it can be over-approximated by a Taylor model (TM) over a bounded time interval. Fig 2 gives an illustration in which the TM $p(\boldsymbol{x}_0, t_0 + \tau) + [a, b]$ is guaranteed to contain the range of the function $\Phi(\boldsymbol{x}_0, t_0 + \tau)$ for any initial state $\boldsymbol{x}_0$ and $t \in [0, \delta]$. In practice, we usually require $\boldsymbol{x}_0$ to be in a bounded set. Such a TM provides a functional over-approximation rather than a pure range over-approximation which allows tracking the dependency from a reachable state to the initial state approximately. Functional over-approximations often can handle more challenging reachability analysis tasks, in which larger initial sets, nonlinear dynamics, or longer time horizons are specified. Recent work has applied interval, polynomial, and TM arithmetic to obtain over-approximations for NNCS evolution [16], [15], [44], [46]. These techniques are often able to compute more accurate flowpipes than the methods in the other group but are often computationally expensive due to the computation of nonlinear multivariate polynomials for

TABLE I: Summary of the tools evaluated in this paper.

| Tool | Category | Plant Dynamics | Activation Function | Set Representation |
|---|---|---|---|---|
| $\alpha, \beta$-CROWN + Flow* | (I) | nonlinear | continuous | Interval and Taylor model |
| NNV | (I) | discrete linear, continuous (CORA) | ReLU, tanh, sigmoid | ImageStar |
| Juliareach | (I) | nonlinear | continuous | Zonotope + Taylor model |
| CORA | (I) | nonlinear | continuous | Polynomial zonotope |
| RINO | (I) | nonlinear | differentiable | interval + interval Taylor series |
| Verisig 2.0 | (II) | nonlinear | differentiable | Taylor model |
| POLAR-Express | (II) | nonlinear | continuous | Taylor model |

tracking the dependencies.

**Existing tools.** We consider the following tools in the experimental evaluation: NNV [18], Verisig 2.0 [44], CORA [48], JuliaReach [49] and RINO [47]. Additionally we also simply combine the use of $\alpha, \beta$-CROWN [40] and Flow* [27] to provide a baseline for the performance of pure range over-approximation. We summarize the key aspects of the tools in Table I. Basically, NNV, CORA, JuliaReach, and RINO compute range over-approximations, while Verisig 2.0 and POLAR-Express compute functional over-approximations.

**Taylor models.** Taylor models were originally proposed to compute higher-order over-approximations for the ranges of continuous functions (see [63]). They can be viewed as a higher-order extension of intervals. A *Taylor model (TM)* is a pair $(p, I)$ wherein $p$ is a polynomial of degree $k$ over a finite group of variables $x_1, \ldots, x_n$ ranging in an interval domain $D \subset \mathbb{R}^n$, and $I$ is the remainder interval. Given a smooth function $f(\boldsymbol{x})$ with $\boldsymbol{x} \in D$ for some interval domain $D$, its TM can be obtained as $(p(\boldsymbol{x}), I)$ such that $p$ is the Taylor expansion of $f$ at some $\boldsymbol{x}_0 \in D$, and $I$ is an interval remainder such that $\forall \boldsymbol{x} \in D.(f(\boldsymbol{x}) \in p(\boldsymbol{x}) + I)$, i.e., $p + I$ is an over-approximation of $f$ at any point in $D$. When the order of $p$ is sufficiently high, the main dependency of the $f$ mapping can be captured in $p$. Basically, the polynomial $p$ can be any polynomial approximation of the function $f$, and it is unnecessary to only use Taylor approximations.

When a function $f(\boldsymbol{x})$ is overly approximated by a TM $(p(\boldsymbol{x}), I)$ w.r.t. a bounded domain $D$, the approximation quality, i.e., size of the overestimation, is directly reflected by the width of $I$, since $f(\boldsymbol{x}) = p(\boldsymbol{x})$ for all $\boldsymbol{x} \in D$ when $I$ is zero by the TM definition. Given two order $k$ TMs $(p_1(\boldsymbol{x}), I_1)$ and $(p_2(\boldsymbol{x}), I_2)$ that are over-approximations of the same function $f(\boldsymbol{x})$ w.r.t. a bounded domain $D \subset \mathbb{R}^n$, we use $(p_1(\boldsymbol{x}), I_1) \prec_k (p_2(\boldsymbol{x}), I_2)$ to denote that the width of $I_1$ is smaller than the width of $I_2$ in all dimensions, i.e., $(p_1(\boldsymbol{x}), I_1)$ is a more accurate over-approximation of $f(\boldsymbol{x})$.

TMs are proven to be powerful over-approximate representations for the flowmap of nonlinear continuous and hybrid systems [64], [65], [66]. Although polynomial zonotopes [67] are also polynomial representations, they are not expressed in the same variables as the system flowmap functions and therefore not functional over-approximations. Interval Taylor Series (ITS) are univariate polynomials in the time variable $t$ where the coefficients are intervals. ITS are often used as nonlinear range over-approximations for ODE solutions [24].

## III. Problem Formulation

We use the formal model presented in Fig 3 to describe the behavior of an NNCS. It is a composition of four modules, each of which models the evolution or input-output mapping of the corresponding component in an NNCS. The top three modules form the controller of the system, it retrieves the sensor data $\boldsymbol{y}$, computes the control input $\boldsymbol{u}$, and applies it to the plant at discrete times $t = 0, \delta_c, \ldots, k\delta_c, \ldots$ for a control step size $\delta_c > 0$. The roles of the modules are described below.

**Plant.** This is a model of the physical process. We use an ODE over the $n$ state variables $\boldsymbol{x}$ and $m$ control inputs $\boldsymbol{u}$ to model the evolution of the physical process such as the movement of a vehicle, the rotation of a DC motor, or the pitch angle change of an aircraft. In the paper, we collectively represent a set of ordered variables $x_1, \ldots, x_n$ by $\boldsymbol{x}$. We only consider the ODEs which are at least locally Lipschitz continuous such that its solution w.r.t. an initial condition $\boldsymbol{x}(0) = \boldsymbol{x}_0 \in \mathbb{R}^n$ is unique [68].

**Preprocessing Module.** This module transforms the sample data. It serves as the gate of the controller. At the time $t = k\delta_c$, for every $k = 0, 1, \ldots$, it retrieves the sensor data $\boldsymbol{y}$, which can be viewed as the image under a mapping from the actual system state $\boldsymbol{x}(t)$, and further transform it to an appropriate format $\boldsymbol{z}$ for the controller's NN component. For instance, a typical preprocessing task in a collision avoidance control system could be computing the relative distances of the moving objects.

**Neural Network.** This is the core computation module for the control inputs. It maps the input data $\boldsymbol{z}$ to the output value $\boldsymbol{v}$ according to the layer-by-layer propagation rule defined in it. In the paper, we only consider *feed-forward neural networks*. Since the paper focuses on the formal verification of NNCSs, the neural network is explicitly defined as a part of the NNCS.

**Postprocessing Module.** This module transforms the NN output value to the control input. Typically, it is used to keep the final control input in the actual actuating range or to filter out inappropriate input values.

We assume that the preprocessing and postprocessing modules can only be defined by a conjunction of *guarded transitions* each of which is in the form of

$$\gamma(\boldsymbol{x}) \;\; \rightarrow \;\; \boldsymbol{x}' = \Pi(\boldsymbol{x}) \tag{1}$$

such that the guard $\gamma(\boldsymbol{x})$ is a conjunction of inequalities in $\boldsymbol{x}$, and $\Pi$ is a transformation from $\boldsymbol{x}$. Here, we assume that all guards are disjoint, and allow (1) to have polynomial arithmetic and the elementary functions $\sin(\cdot)$, $\cos(\cdot)$, $\exp(\cdot)$, $\log(\cdot)$, $\sqrt{\cdot}$. Then the expressiveness is sufficient to define lookup tables.

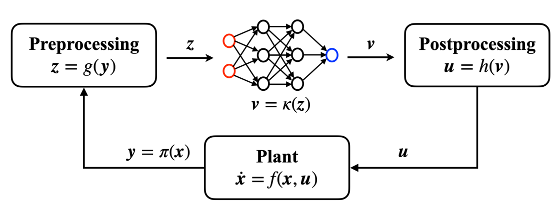
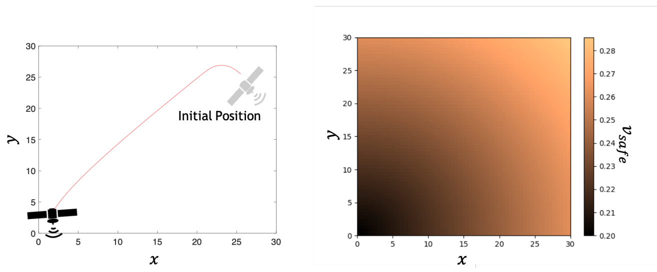



Fig. 3: Formal model of NNCS.




(a) (b)

Fig. 4: The 2D spacecraft docking environment. (a) The goal is to start from some initial position and move towards the origin. (b) The safe speed limit $v_{safe}$ decreases as the position of the spacecraft approaches the origin.

**Example 1** (2D Spacecraft Docking)**.** *We consider the docking of a spacecraft in a 2D plane. The benchmark is described in [69]. As shown in Fig 4, the control goal is to steer the spacecraft to the position at the origin while the velocity should be kept in a safe range. The whole benchmark can be modeled by an NNCS with 5 variables:* $\boldsymbol{x} = (x, y, v_x, v_y, v_{safe})^T$ *wherein* $(x, y)$ *denotes the position of the spacecraft,* $(v_x, v_y)$ *denotes the velocity, and* $v_{safe} = 0.2 + 0.002054\sqrt{x^2 + y^2}$ *is a particular variable that indicates a position-dependent safe limit on the speed. The dynamics is defined as in equation 2 where* $f_x$ *and* $f_y$ *constitute the control input* $\boldsymbol{u} = (f_x, f_y)$ *which is obtained by a neural network controller* $\kappa$*. The input* $\boldsymbol{z} = (\frac{x}{1000}, \frac{y}{1000}, 2v_x, 2v_y, \sqrt{v_x^2 + v_y^2}, v_{safe})$ *of the neural network is prepossessed from* $\boldsymbol{x}$*. The output* $\boldsymbol{v} = (u_x, u_y)$ *of the neural network is postprocessed to* $f_x = \tanh(u_x), f_y = \tanh(u_y)$*.*

$$\begin{aligned} &\dot{x} = v_x, \quad \dot{y} = v_y, \quad \dot{v}_{safe} = \frac{0.002054(x \cdot v_x + y \cdot v_y)}{v_{safe}} \\ &\dot{v}_x = 0.002054 v_y + 3 \times (0.001027)^2 x + f_x/12, \\ &\dot{v}_y = -0.002054 v_x + f_y/12, \end{aligned} \tag{2}$$

**Executions of NNCSs.** Starting from an initial state $\boldsymbol{x}_0 \in \mathbb{R}^n$, for all $i = 0, 1, \ldots$, the system state $\boldsymbol{x}(t)$ in the $(i+1)$-st control step $t \in [i\delta_c, (i+1)\delta_c]$ is defined by the solution of the ODE $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}_i)$ w.r.t. the initial state $\boldsymbol{x}(i\delta_c)$ and the control input $\boldsymbol{u}_i$ which is obtained as $\boldsymbol{u}_i = h \circ \kappa \circ g \circ \pi(\boldsymbol{x}(i\delta_c))$, where $h, \kappa, g, \pi$ are shown in Fig 3. If we denote the solution of the ODE w.r.t an initial state $\boldsymbol{x}_0$ and a particular control input $\boldsymbol{u}'$ by $\boldsymbol{x}(t) = \Phi_f(\boldsymbol{x}_0, t, \boldsymbol{u}')$, the system state at a time $t \in [i\delta_c, (i+1)\delta_c]$ for any $i \geq 0$ from the initial state $\boldsymbol{x}_0$ can be expressed recursively by

$$\begin{aligned} &\boldsymbol{x}(t) = \Phi_f(\boldsymbol{x}(i\delta_c), t - i\delta_c, \boldsymbol{u}_i) \\ &\text{where } \boldsymbol{u}_i = h \circ \kappa \circ g \circ \pi(\boldsymbol{x}(i\delta_c)) \end{aligned} \tag{3}$$

such that $\boldsymbol{x}(0) = \boldsymbol{x}_0$. We also call this state a *reachable state*. Without noises or disturbances from the environment, an NNCS has deterministic behavior, and its evolution can be defined by a *flowmap* function in the form of $\Phi(\boldsymbol{x}_0, t)$, i.e., the reachable state from an initial state $\boldsymbol{x}_0$ at a time $t$ is $\boldsymbol{x}(t) = \Phi(\boldsymbol{x}_0, t)$, and it is *uniquely determined by the initial state and the time*. Unfortunately, $\Phi$ usually does not have a closed-form expression.

**Algorithm 2:** Framework of POLAR-Express.

**Input:** Plant dynamics $\boldsymbol{x}' = f(\boldsymbol{x}, \boldsymbol{u})$, preprocessing $g(\cdot)$, postprocessing $h(\cdot)$, NN controller $\kappa(\cdot)$, number of control steps $K$, initial set $X_0$.

**Output:** Over-approximation of the reachable set over the time interval of $[0, K\delta_c]$ with $\delta_c$ as control step size.

```
R ← ∅, X_0 ← X_0;
for i = 0 to K − 1 do
    Computing a superset Y_i for the range of π(X_i);
    Computing a superset Z_i for the range of g(Y_i);
    Computing a superset V_i for the range of κ(Z_i), with
      multi-threading support;
    Computing a superset U_i for the range of h(V_i);
    Computing a set F of flowpipes for the continuous
      dynamics ẋ = f(x, u) with u ∈ U_i from the initial set
      x(0) ∈ X_i over the time interval of [iδ_c, (i + 1)δ_c];
    R ← R ∪ F;
    Evaluating an over-approximation for the reachable set
      at the time t = (i + 1)δ_c based on F and assigning it
      to X_{i+1};
end for
return R.
```

The reachability analysis task with respect to two given system states $\boldsymbol{x}, \boldsymbol{x}'$ and an NNCS asks whether $\boldsymbol{x}'$ is reachable from $\boldsymbol{x}$ under the evolution of the system. Reachability analysis plays a key role in the safety verification of dynamic systems. However, it is a notoriously difficult task due to the undecidability of the reachability problem on the systems defined by nonlinear difference equations [70]. In order to prove the safety of a system, most of the reachability techniques seek to compute an over-approximation of the reachable set. If no unsafe state is contained in the over-approximated reachable set, the system is guaranteed to be safe.

## IV. Framework of POLAR-Express

We present the POLAR-Express framework in Algorithm 2 to compute flowpipes for NNCSs. It uses the standard set-propagation framework Algorithm 1 but has the following novel elements: *(1) Polynomial over-approximation for activation functions using* Bézier curves*. (2) Symbolic representation of TM remainders in layer-by-layer propagation. (3) A seamless integration of the above techniques to compute accurate flowpipes for NNCSs.* The last two novel elements are the new contributions of POLAR-express when compared to POLAR: *(4) A more precise and efficient method for propagating TMs across ReLU activation. (5) Multi-threading support to parallelize the computation in the layer-by-layer propagation of TMs for NN.* The details are explained below.

**Computing $\mathcal{Y}_i$ and $\mathcal{U}_i$.** Given a preprocessing or postprocessing module and its input set which is represented as a TM, an output TM can be obtained by computing the reachable sets of the guarded transitions. Given a guarded transition of the form (1) along with a TM $S$ for $\boldsymbol{x}$'s range, the reachable set $S'$, i.e., the range of $\boldsymbol{x}'$, can be computed by first computing the intersection $S_I = S \cap \{\boldsymbol{x} \mid \gamma(\boldsymbol{x})\}$ and then evaluating

**Algorithm 3:** Layer-by-layer propagation using polynomial arithmetic and TMs

**Input:** Input TM $(p_1(\boldsymbol{x}_0), I_1)$ with $\boldsymbol{x}_0 \in X_0$, the $M+1$ matrices $W_1, \ldots, W_{M+1}$ of the weights on the incoming edges of the hidden and the output layers, the $M+1$ vectors $B_1, \ldots, B_{M+1}$ of the neurons' bias in the hidden and the output layers.

**Output:** a TM $(p_r(\boldsymbol{x}_0), I_r)$ that contains the set $\kappa((p_1(\boldsymbol{x}_0), I_1))$.

1: $(p_r, I_r) \leftarrow (p_1, I_1)$;
2: **for** $i = 1$ to $M+1$ **do**
3: $\quad (p_t, I_t) \leftarrow W_i \cdot (p_r, I_r) + B_i$;
4: $\quad$ Computing a polynomial approximation $p_{\sigma,i}$ for the vector of the current layer's activation functions $\sigma$ w.r.t. the domain $(p_t, I_t)$;
5: $\quad$ Evaluating a conservative remainder $I_{\sigma,i}$ for $p_{\sigma,i}$ w.r.t. the domain $(p_t, I_t)$;
6: $\quad (p_r, I_r) \leftarrow p_{\sigma,i}(p_t + I_t) + I_{\sigma,i}$;
7: **end for**
8: **return** $(p_r, I_r)$.

$\Pi(S_I)$ using TM arithmetic [71]. Although TMs are not closed under an intersection with a semi-algebraic set, we may use the domain contraction method proposed in [64] to derive an over-approximate TM for the intersection.

### *A. Layer-by-Layer Propagation using TMs*

POLAR-Express uses the layer-by-layer propagation scheme to compute a TM output for the NN, and features the following key novelties: (a) A method to selectively compute Taylor or Bernstein polynomials for activation functions. The purpose is to *derive a smaller error according to the approximated function and its domain*. The Bernstein polynomials are represented in their *Bézier forms*. (b) A technique to symbolically represent the intermediate linear transformations of TM interval remainders during the layer-by-layer propagation. The purpose of using Symbolic Remainders **(SR)** is to *reduce the accumulation of overestimation produced in composing a sequence of TMs*. The approach is described as follows.

**Layer-by-Layer Propagation with Multi-Threading Support**. The framework of layer-by-layer propagation has been widely used to compute NN output ranges. Most of the existing methods use range over-approximations such as intervals with constant bounds [57], [72] or linear polynomial bounds [38], zonotopes [62], [39], star sets [18]. Some TM-based methods are also proposed [16], [15], [45] to obtain functional over-approximations for the input-output mapping of an NN. However, the use of functional over-approximations in the reachability analysis of an NNCS as a whole has not been well investigated. Hence, we propose the following approach to better over-approximate the input-output dependency than the existing state-of-the-art.

Algorithm 3 presents the main framework of our approach without using SR and focuses on the novelty coming from the tighter TM over-approximation for the activation functions (Lines 4 and 5). Before introducing our selective over-approximation method, we describe how a TM output is computed from a given TM input for a single layer. The idea is illustrated in Fig. 5. The circles in the right column denote the neurons in the current layer which is the $(i+1)$-th layer, and those in the left column denote the neurons in the previous layer. The weights on the incoming edges to the current layer are organized as a matrix $W_i$, while we use $B_i$ to denote the vector organization of the biases in the current layer. Given that the output range of the neurons in the previous layer is represented as a TM (vector) $(p_i(\boldsymbol{x}_0), I_i)$ where $\boldsymbol{x}_0$ are the variables ranging in the NNCS initial set. Then, the output TM $(p_{i+1}(\boldsymbol{x}_0), I_{i+1})$ of the current layer can be obtained as follows. First, we compute the polynomial approximations $p_{\sigma_1,i}, \ldots, p_{\sigma_l,i}$ for the activation functions $\sigma_1, \ldots, \sigma_l$ of the neurons in the current layer. Second, interval remainders $I_{\sigma_1,i}, \ldots, I_{\sigma_l,i}$ are evaluated for those polynomials to ensure that for each $j = 1, \ldots, l$, $(p_{\sigma_j,i}, I_{\sigma_j,i})$ is a TM of the activation function $\sigma_j$ w.r.t. $z_j$ ranging in the $j$-th dimension of the set $W_i(p_i(\boldsymbol{x}_0) + I_i)$. Third, $(p_{i+1}(\boldsymbol{x}_0, I_{i+1}))$ is computed as the TM composition $p_{\sigma,i}(W_i(p_i(\boldsymbol{x}_0)+I_i)+I_{\sigma,i}$ where $p_{\sigma,i}(\boldsymbol{z}) = (p_{\sigma_1,i}(z_1), \ldots, p_{\sigma_l,i}(z_k))^T$ and $I_{\sigma,i} = (I_{\sigma_1,i}, \ldots, I_{\sigma_l,i})^T$. Hence, when there are multiple layers, starting from the first layer, the output TM of a layer is treated as the input TM of the next layer, and the final output TM is computed by composing TMs layer by layer. Besides, we use $(p_{j,i}, I_{j,i})$ for $j = 1, \ldots, l$ to represent the TMs associated with the $l$ neurons in a linear layer. The computation of those TMs can be conducted in parallel. So is the propagation through the activation functions in a layer. POLAR-Express realizes such parallelism via multi-threading to retain time efficiency when the dimension of the NN layers is large.

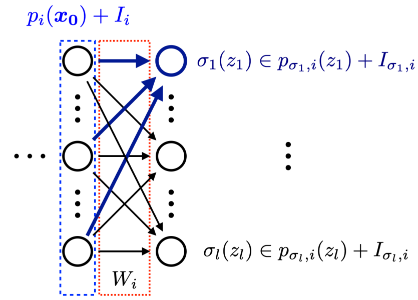



Fig. 5: Single layer propagation

**Polynomial Approximations to TMs.** Basically, a TM only defines an over-approximate mapping and is independent of the approximation method used for the polynomial part. Thus, we consider using both Taylor and Bernstein approximations when propagating through an activation function and choose the one that produces less overestimation after a TM combination. The following example shows that the selection cannot be determined only based on the approximation error. Given the TMs $(p_1, I_1)$, $(p_2, I_2)$ which are both TM over-approximations for the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ w.r.t. a TM domain $x \in q(y) + J$:

$$(p_1, I_1) = (0.5 + 0.25x - 0.02083x^3, \text{[-7.93e-5, 1.92e-4]})$$
$$(p_2, I_2) = (0.5 + 0.24855x - 0.004583x^3, \text{[-2.42e-4, 2.42e-4]})$$
$$(q, J) = (0.1y - 0.1y^2, \text{[-0.1,0.1]}).$$

where $y \in [-1, 1]$. However the composition $(p_1(q(y) + J) + I_1)$ produces a TM with the remainder $[-0.0466, 0.0477]$, while the remainder produces by $p_2(q(y) + J) + I_2$ is $[-0.0253, 0.0253]$ which is smaller. In other words, a smaller polynomial approximation error does not always lead to a smaller error in combination. Therefore, it motivates us to do a selection after the combination. We generalize this phenomenon by defining the *accuracy preservation problem*, and obviously, the answer is **No** if TM arithmetic is used.

**Definition 2** (Accuracy preservation problem)**.** *If both $(p_1(\boldsymbol{x}), I_1)$ and $(p_2(\boldsymbol{x}), I_2)$ are over-approximations of $f(\boldsymbol{x})$ with $\boldsymbol{x} \in D$, and $(p_1(\boldsymbol{x}), I_1) \prec_k (p_2(\boldsymbol{x}), I_2)$. Given another function $g(\boldsymbol{y})$ which is already over-approximated by a TM $(q(\boldsymbol{y}), J)$ whose range is contained in $D$. Then,* does $p_1(q(\boldsymbol{y})+J)+I_1 \prec_k p_2(q(\boldsymbol{y})+J)+I_2$ still hold using order $k$ TM arithmetic?

### *B. Bernstein Over-approximation for Activation Functions*

Now we turn to our Bernstein over-approximation method for activation functions. It first computes a Bernstein polynomial for the function and then evaluates a remainder interval to ensure the over-approximation. The polynomials are in the Bézier form.

**Definition 3** (Bernstein polynomial)**.** *Given a continuous function $f(x)$ with $x \in [a, b]$, its order $k$ Bernstein polynomial $p_k(x)$ is defined by*

$$\sum_{i=0}^{n} \left( f\left(a + \frac{i}{k}(b-a)\right) \binom{k}{i} \left(\frac{x-a}{b-a}\right)^i \left(\frac{b-x}{b-a}\right)^{k-i} \right) \tag{4}$$

**Bernstein approximation in Bézier form.** The use of Bernstein approximation only requires the activation function to be continuous in $(p_t, I_t)$ and can be used not only in more general situations but also to obtain better polynomial approximations than Taylor expansions (see [73]). We first give a general method to obtain a Bernstein over-approximation for an arbitrary continuous function, and then present *a more accurate approach only for ReLU functions*. Given that the activation functions in a layer are collectively represented as a vector $\sigma(\boldsymbol{z})$ and $\boldsymbol{z}$ ranges in a TM $(p_t, I_t)$. Then the order $k$ Bernstein polynomial $p_{\sigma_j,i}(z_j)$ for the activation function $\sigma_j$ of the $j$-th neuron. It can be computed as (4) while $f$ is $\sigma_j$, $a, b$ are the lower and upper bound respectively of the range in the $j$-th dimension of $(p_t, I_t)$, and they can be obtained from an interval evaluation of the TM.

**Remainder Evaluation.** The remainder $I_{\sigma_j,i}$ for the polynomial $p_{\sigma_j,i}$ can be obtain as a symmetric interval $[-\epsilon_j, \epsilon_j]$ such that $\epsilon_j$ is

$$\max_{s=1,\cdots,m} \left( \left| p_{\sigma_j,i}\left(\frac{\overline{Z}_j - \underline{Z}_j}{m}(s-\frac{1}{2}) + \underline{Z}_j\right) - \sigma_j\left(\frac{\overline{Z}_j - \underline{Z}_j}{m}(s - \frac{1}{2}) + \underline{Z}_j\right) \right| + L_j \cdot \frac{\overline{Z}_j - \underline{Z}_j}{m} \right)$$

wherein $L_j$ is a Lipschitz constant of $\sigma_j$ with the domain $(p_t, I_t)$, and $m$ is the number of samples that are uniformly selected to estimate the remainder. The soundness of the error bound estimation above is proven in [15] for multivariate Bernstein polynomials. Since the univariate Bernstein polynomial, which we use in this paper, is a special case of multivariate Bernstein polynomials, our approach is also sound.

**More Precise and Efficient Bernstein Over-approximation for ReLU**. The above Bernstein over-approximation method works on all continuous activation functions, however, if a function is convex or concave on the domain of interest, a more accurate Bernstein over-approximation that is represented as a TM can be obtained as follows. Given a continuous function $f(x)$ with $x \in D$ such that $f$ is convex on the domain, the Bernstein polynomials of $f$ are no smaller than $f$ at any point in $D$. Thus, a tight upper bound for $f$ can be computed as one of its Bernstein polynomial $p$, while a tight lower bound can be obtained by moving $p$ straight down by the distance which is equivalent to the maximum difference of $f$ and $p$ for $x \in D$. When $f$ is ReLU and $0 \in D$, it is convex on $D$, and its maximum difference to any Bernstein polynomial $p$ is $p(0)$. This direct over-approximation avoids sampling for error estimation and thus is more precise and efficient than [15]. We give the particular Bernstein over-approximation method for ReLU functions by Algorithm 4. An example is illustrated in Fig. 7, and the effectiveness of this approach is shown in example 2.

**Lemma 1.** *Given that $p_k(x)$ is the order $k \geq 1$ Bernstein polynomial of a convex function $f(x)$ with $x \in [a, b]$. For all $x \in [a, b]$, we have that (i) $f(x) \leq p_k(x)$ and (ii) $p_{k+1}(x) \leq p_k(x)$.*

*Proof.* The Lemma is proven in [74] for the domain $x \in [0, 1]$. However, it also holds on an arbitrary domain $x \in [a, b]$ after we replace the lower and upper bounds in the Bernstein polynomials by $a$ and $b$. □

**Corollary 1.** *If $p(x)$ is the order $k \geq 1$ Bernstein polynomial of ReLU$(x)$ with $x \in [a, b]$, then $0 \leq$ ReLU$(x) \leq p(x)$ for all $x \in [a, b]$.*

**Lemma 2.** *Given that $p(x)$ is the order $k \geq 1$ Bernstein polynomial of ReLU$(x)$ with $x \in [a, b]$ such that $a < 0 < b$, then we have that $p(x) -$ ReLU$(x) \leq p(0)$ for all $x \in [a, b]$.*

*Proof.* Since ReLU$(x)$ is convex over the domain, by [74], so is $p(x)$. Therefore, the second derivative of $p$ w.r.t. $x$ is non-negative. By evaluating the first derivatives of $p$ at $x = a$ and $x = b$, we have that $\frac{dp}{dx}|_{x=a} \geq 0$ and $\frac{dp}{dx}|_{x=b} \leq 1$. Since the first derivatives of ReLU$(x)$ are 0 and 1 when $x \in [a, 0)$ and $x \in (0, b]$ (ReLU(0) is continuous but not differentiable) respectively, $p(a) = \text{ReLU}(a)$, and $p(b) = \text{ReLU}(b)$, we have that the function $p(x) - \text{ReLU}(x)$ monotonically increasing when $x \in [a, 0)$ and decreasing when $x \in (0, b]$, and $p(x) - \text{ReLU}(x)$ is continuous at $x = 0$, hence its maximum value is given by $p(0)$. □

**Example 2.** *In Fig 6, an NNCS with ReLU as the activation functions starts from an initial set near $(0.4, 0.45)$ and moves towards a target set enclosed by the yellow rectangle (see*

**Algorithm 4:** Efficient and Tight Bernstein over-approximation for ReLU functions

**Input:** Domain $D = [a, b]$ $(a \leq b)$ of the ReLU function, approximation order $k$.
**Output:** a TM that overly approximates the ReLU on $D$.

**if** $a \geq 0$ **then return** $(x, [0, 0])$;
**else if** $b \leq 0$ **then return** $(0, [0, 0])$;
**else**
    Computing the order $k$ Bernstein polynomial $p$;
    $\varepsilon \leftarrow p(0)$;
    **return** $(p - 0.5\varepsilon, [-0.5\varepsilon, 0.5\varepsilon])$;

*details in benchmark 5 [16], [15]). POLAR-Express with this new precise and efficient over-approximation approach for the ReLU function generates tighter flowpipes than POLAR. The runtime of POLAR-Express is* $1.8s$ *while the runtime of POLAR is* $7.1s$. *This result serves as evidence that this novel Bernstein over-approximation for ReLU achieves better verification efficiency and accuracy.*

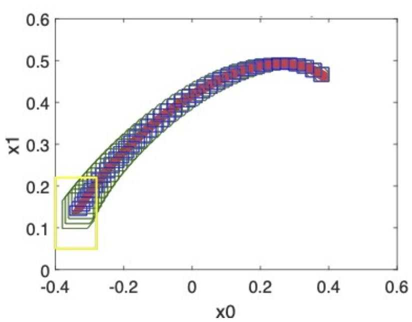



Fig. 6: POLAR-Express (blue) generates tighter over-approximations than POLAR (green) under the same hyper-parameters; the red curves are the simulated traces; the yellow rectangle is the target set.

## C. Using Symbolic Remainders

A main source of overestimation in interval arithmetic is the computation of linear mappings. Given a box (Cartesian product of single intervals) $I$, its image under a linear mapping $\boldsymbol{x} \mapsto A\boldsymbol{x}$ is often not a box and has to be over-approximated by a box in interval arithmetic. For a sequence of linear mappings, the resulting box is often unnecessarily large due to the overestimation accumulated in each mapping, known as the *wrapping effect* [55]. To avoid this class of overestimation, we may symbolically represent the intermediate boxes and only do an interval evaluation at last. For example, if we need to compute the image of the box $I$ through the linear mappings: $\boldsymbol{x} \mapsto A_1\boldsymbol{x}, \ldots, \boldsymbol{x} \mapsto A_m\boldsymbol{x}$, the box $I$ is kept symbolically and the composite mapping is computed as $A' = A_m \cdots A_1$. A tight interval enclosure for the image can be obtained from evaluating $A'I$ using interval arithmetic.

Although TM arithmetic uses polynomials to symbolically represent the variable dependencies, it is not free from wrapping effects since the remainder is always computed using

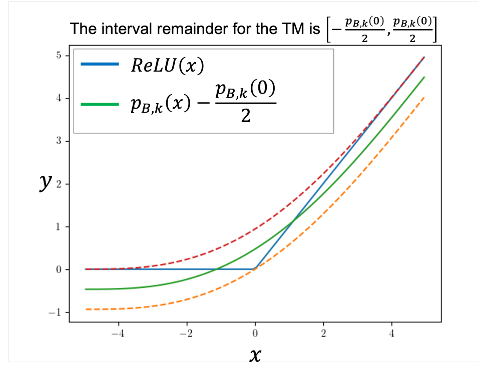



Fig. 7: The TM over-approximation $p(x) + I$ of $\text{ReLU}(x)$ is given by $p(x) = p_{B,k}(x) - \frac{p_{B,k}(0)}{2}$ and $I = [-\frac{p_{B,k}(0)}{2}, \frac{p_{B,k}(0)}{2}])$ where $p_{B,k}(0)$ is the Bernstein polynomial $p_{B,k}(x)$ evaluated at $x = 0$. As shown, for $x \in [a, b](a < 0 < b)$, the bounds of the interval remainder $I$ are tight for any order $k \geq 1$ Bernstein polynomials approximation.

interval arithmetic. Consider the TM composition for computing the output TM of a single layer in Fig. 5, the output TM $p_{\sigma,i}(W_i(p_i(\boldsymbol{x}_0) + I_i) + B_i) + I_{\sigma,i}$ equals to $Q_iW_ip_i(\boldsymbol{x}_0) + Q_iW_iI_i + Q_iB_i + p^R_{\sigma,i}(W_i(p_i(\boldsymbol{x}_0) + I_i) + B_i) + I_{\sigma,i}$ such that $Q_i$ is the matrix of the linear coefficients in $p_{\sigma,i}$, and $p^R_{\sigma,i}$ consists of the terms in $p_{\sigma,i}$ of the degrees $\neq 1$. Therefore, the remainder $I_i$ in the second term can be kept symbolically such that we do not compute $Q_iW_iI_i$ out as an interval but keep its transformation matrix $Q_iW_i$ to the subsequent layers. Given the image $S$ of an interval under a linear mapping, we use $\underline{S}$ to denote that it is kept symbolically, i.e., we keep the interval along with the transformation matrix, and $\overline{S}$ to denote that the image is evaluated as an interval.

Next, we present the use of SR in layer-by-layer propagation. Starting from the NN input TM $(p_1(\boldsymbol{x}_0), I_1)$, the output TM of the first layer is computed as

$$\underbrace{Q_1W_1p_1(\boldsymbol{x}_0) + Q_1B_1 + p^R_{\sigma,1}(W_1(p_1(\boldsymbol{x}_0) + I_1) + B_1) + I_{\sigma,1}}_{q_1(\boldsymbol{x}_0)+J_1}$$
$$+ \underline{Q_1W_1I_1},$$

which can be kept in the form of $q_1(\boldsymbol{x}_0) + J_1 + \underline{Q_1W_1I_1}$. Using it as the input TM of the second layer, we have the following TM

$$\begin{aligned}
&p_{\sigma,2}(W_2(q_1(\boldsymbol{x}_0) + J_1 + \underline{Q_1W_1I_1}) + B_2) + I_{\sigma,2} \\
=&\underbrace{Q_2W_2q_1(\boldsymbol{x}_0) + Q_2B_2 + p^R_{\sigma,2}(W_2(q_1(\boldsymbol{x}_0) + J_1 + \overline{Q_1W_1I_1}) + B_2) + I_{\sigma,2}}_{q_2(\boldsymbol{x}_0)+J_2} \\
&+ \underline{Q_2W_2J_1} + \underline{Q_2W_2Q_1W_1I_1}
\end{aligned}$$

for the output range of the second layer. Therefore the output TM of the $i$-th layer can be obtained as $q_i(\boldsymbol{x}_0) + \mathbb{J}_i + \underline{Q_iW_i \cdots Q_1W_1I_1}$ such that $\mathbb{J}_i = J_i + \underline{Q_iW_iJ_{i-1}} + \underline{Q_iW_iQ_{i-1}W_{i-1}J_{i-2}} + \cdots + \underline{Q_iW_i \cdots Q_2W_2J_1}$.

We present the SR method in Algorithm 5 where we use two lists: $\mathcal{Q}[j]$ for $Q_iW_i \cdot \cdots \cdot Q_jW_j$ and $\mathcal{J}[j]$ for $\mathbb{J}_j$ to keep the intervals and their linear transformations. The symbolic remainder representation is replaced by its interval enclosure $I_r$ at the end of the algorithm.

**Algorithm 5:** TM output computation using symbolic remainders, input and output are the same as those in Algorithm 3

```
Setting 𝒬 as an empty array which can keep M + 1 matrices;
Setting 𝒥 as an empty array which can keep M + 1 multidimensional intervals, 𝕁 ← 0;
for i = 1 to M + 1 do
    Computing the composite function p_{σ,i} and the remainder interval I_{σ,i} using the BP technique;
    Evaluating q_i(x_0) + J_i based on 𝕁 and 𝒬[1]I_1;
    𝕁 ← J_i and Φ_i ← Q_i W_i;
    for j = 1 to i − 1 do 𝒬[j] ← Φ_i · 𝒬[j]; end for
    Adding Φ_i to 𝒬 as the last element;
    for j = 2 to i do 𝕁 ← 𝕁 + 𝒬[j] · 𝒥[j − 1]; end for
    Adding J_i to 𝒥 as the last element;
end for
Computing an interval enclosure I_r for 𝕁 + 𝒬[1]I_1;
return q_{M+1}(x_0) + I_r.
```

**Time and space complexity.** POLAR-Express and POLAR have similar time and space complexity. Although Algorithm 5 produces TMs with tighter remainders than Algorithm 3, because of the symbolic interval representations under linear mappings, it requires (1) two extra arrays to keep the intermediate matrices and remainder intervals, and (2) two extra inner loops that perform $i-1$ and $i-2$ iterations in the $i$-th outer iteration. The size of $Q_iW_i \cdot \dots \cdot Q_jW_j$ is determined by the rows in $Q_i$ and the columns in $W_j$, and hence the maximum number of neurons in a layer determines the maximum size of the matrices in $\mathcal{Q}$. Similarly, the maximum dimension of $J_i$ is bounded by the maximum number of neurons in a layer. Because of the two inner loops, the time complexity of Algorithm 5 is quadratic in $M$, whereas Algorithm 3 is linear in $M$.

**Theorem 1.** *In Algorithm 2, if $(p(\boldsymbol{x}_0,\tau), I)$ is the $i$-th TM flowpipe computed in the $j$-st control step, then for any initial state $\boldsymbol{c} \in X_0$, the box $p(\boldsymbol{c},\tau) + I = p'(\tau) + I$ contains the actual reachable state $\varphi_{\mathcal{N}}(\boldsymbol{c}, (j-1)\delta_c + (i-1)\delta + \tau)$ for all $\tau \in [0, \delta]$.*

## V. Benchmark Evaluations

We conduct a comprehensive comparison with state-of-the-art tools across a diverse set of benchmarks. In addition, we discuss in detail the applicability and comparative advantages of different techniques. The experiments were performed on a machine with a 6-core 2.20 GHz Intel i7 CPU and 16GB of RAM. For tools that can leverage GPU acceleration such as $\alpha,\beta$-CROWN, the experiments were run with the aid of an Nvidia GeForce GTX 1050Ti GPU. The multi-threading support is realized by using the C++ standard library. Considering the overhead introduced by multi-threading, we also measure the performance of the application under a single thread to identify the bottleneck caused by multi-threading.

**Benchmarks.** Our NNCS benchmark suite consists of Benchmarks 1-6 [16], [15], Discrete Mountain Car (MC) [17], Adaptive Cruise Control (ACC) [18], 2D Spacecraft Docking [69], Attitude Control [46], Quadrotor-MPC [17] and QUAD20 [46]. Benchmark 6, Attitude Control, and QUAD20 are directly used or incorporated with minor modifications in recent ARCH-COMP AINNCS competitions [75]. The remaining examples either do not appear in ARCH-COMP or are substantially different from the mentioned instances (e.g., ACC is 8-dimensional in this paper, whereas 6-dimensional in ARCH-COMP). All the benchmarks are online at https://github.com/ChaoHuang2018/POLAR_Tool/tree/main/POLAR_Express_Benchmarks, where the dynamics and NN in each benchmark are formatted in the languages required by the tools. The details of the benchmarks are as follows.

TABLE II: Reachability verification tasks for Benchmark 1-6.

| Benchmark | Initial set | Target set[1] | $N$ |
|---|---|---|---|
| 1 | $[0.8,0.9]\times[0.5,0.6]$ | $[0,0.2]\times[0.05,0.3]$ | 35 |
| 2 | $[0.7,0.9]\times[0.7,0.9]$ | $[-0.3,0.1]\times[-0.35,0.5]$ | 10 |
| 3 | $[0.8,0.9]\times[0.4,0.5]$ | $[0.2,0.3]\times[-0.3,-0.05]$ | 60 |
| 4 | $[0.25,2.27]\times[0.08,0.1]$ $\times[0.25,0.27]$ | $[-0.2,-0.1]\times[0,0.05]$ | 10 |
| 5 | $[0.38,0.4]\times[0.45,0.47]$ $\times[0.25,0.27]$ | $[-0.43,-0.15]\times[0.05,0.22]$ | 10 |
| 6 | $[-0.77,-0.75]\times[-0.45,-0.43]$ $\times[0.51,0.54]\times[-0.3,-0.28]$ | $[-0.1,0.2]\times[-0.9,-0.6]$ | 10 |

*Benchmark 1-6.* The reachability verification task asks if the NNCS can reach the target set from any initial state in $N$ control steps, as in Table II.

*Discrete-Time Mountain Car (MC).* It is a 2-dimensional NNCS describing an underpowered car driving up a steep hill. We consider the initial set defined by $x_0 \in [-0.53, -0.5]$ and $x_1 = 0$. The target is $x_0 \geq 0.2$ and $x_1 \geq 0$ where the car reaches the top of the hill and is moving forward. The total control steps $N$ is 150.

*Adaptive Cruise Control (ACC).* The benchmark models the moves of a lead vehicle and an ego vehicle. The NN controller tries to maintain a safe distance between them. We use the definition of the initial and target set given in [45], and the number of control steps is $N = 50$.

*2D Spacecraft Docking (Example 1).* The initial set is defined by $x, y \in [24, 26]$, $v_x = v_y = -0.1378$, and $v_{safe} \in [0.2697, 0.2755]$ which is directly computed based on the ranges of $x, y$. The total control steps $N$ is 120. In this benchmark, we only verify the safety property. That is, the NN controller should maintain $\sqrt{v_x^2 + v_y^2} \leq v_{safe}$ all the time.

*Attitude Control & QUAD20.* The reachability problems for the two benchmarks are the same as the ones given in [46].

*Quadrotor-MPC.* This benchmark is originally given in [45]. It consists of a quadrotor and a planner. The position of the quadrotor is indicated by the state variables $(p_x, p_y, p_z)$, while the velocity in the 3 dimensions is given by $(v_x, v_y, v_z)$. The velocity of the planner is $(b_x, b_y, b_z)$ which has a piecewise-constant definition: $(b_x, b_y, b_z) = (-0.25, 0.25, 0)$ for $t \in [0, 2]$ (first 10 steps), $(b_x, b_y, b_z) = (-0.25, -0.25, 0)$ for $t \in [2, 4]$, $(b_x, b_y, b_z) = (0, 0.25, 0)$ for $t \in [4, 5]$, and

[1]The target set is defined on the first two dimensions by default.

$(b_x, b_y, b_z) = (0.25, -0.25, 0)$ for $t \in [5, 6]$. The control input $\theta$, $\phi$, $\tau$ is determined by an NN "bang-bang" controller, which is a classifier mapping system states to a finite set of control actions. The initial set is defined as $p_x - q_x \in [-0.05, -0.025]$, $p_y - q_y \in [-0.025, 0]$, and $p_z - q_z = v_x = v_y = v_z = 0$. The verification task asks to prove that all reachable states in 30 control steps should be in the safe set $-0.32 \leq p_x - q_x, p_y - q_y, p_z - q_z \leq 0.32$.

**Evaluation Metrics.** The tools are compared based on their performance on all of the benchmarks. Since tools use different hyper-parameters, we tune their settings for each benchmark and try to make them produce similar reachable set over-approximations and compare the time costs. For a tool that is not able to handle a benchmark, we present its result with the best setting that we can find. Those hyper-parameters can also be found in the Github collection.

**Stopping Criteria.** We stop the run of a tool when the reachability problem is proven, or the tool raises an error or is terminated by the operating system due to a runtime system error such as being out of memory. Hence, every verification produces one of the following four results: **(Yes)** the reachability property is proven, **(No)** the reachability property is disproved, **(U)**nknown the computed over-approximation is too large to prove or disprove the property with the best tool setting we can find, **(DNF)** (did not finish) a tool or system error is reported and the reachability computation fails.

**Experimental Results.** Table III and Fig. 8 show the results. Because Quadrotor-MPC is a hybrid system, only POLAR-Express, Verisig 2.0, and $\alpha, \beta$-CROWN + Flow* are able to deal with it. According to CORA's manual, the tool does not have an API to directly model hybrid systems with NN controllers. It is verified to be safe by POLAR-Express with 13.1 seconds and by Verisig 2.0 with 961.4 seconds. The result is Unknown for $\alpha, \beta$-CROWN + Flow* after 10854 seconds.

### A. Challenges with Running Other Tools

We found soundness issues when running the CORA tool for Benchmark 5 and the QUAD20 example. In Benchmark 5, both simulation traces and reachable sets of CORA deviate from the other tools. In QUAD20, the reachable sets computed by CORA cannot cover the simulation traces (i.e. not an over-approximation). We contacted the authors of CORA and they confirmed that this issue is caused an internal bug. We fixed it and updated the results in Table III.

The default setup in JuliaReach reports the runtime of running the *same* reachable set computation the second time after a "warm-up" run (whose runtime is not included), likely to take advantage of cache effects or saved computations. For a fair comparison with the other tools, we report the runtime of the first run.

RINO has three "DNF"s caused by division-by-zero errors. Both RINO and JuliaReach have their own plot functions, while the other tools plot reachable sets in MATLAB. This is an issue in several examples such as the plot function in RINO taking too long to plot the reachable sets.

For the most complicated QUAD20 example, Verisig 2.0 failed after 17939 seconds during 1-step reachable set computation, $\alpha, \beta$-CROWN + Flow* failed after 3 control steps, NNV failed after 1 step, CORA failed after 20 steps, the reachable set of JuliaReach explodes after 25 control steps, and RINO has the division-by-zero error.

### B. Experimental Comparison and Discussion

According to the experimental results in Table III and Fig 8, POLAR-Express can verify all the benchmarks and achieve the overall best performance in terms of the tightness of the reachable set computations and runtime efficiency. We can also observe that the multi-threading support for POLAR-Express helps to reduce runtime in higher dimensional systems. However, because of the overhead involved, it may take longer than the single-threaded version for some of the lower dimensional benchmarks. Thus, we believe that multi-threading support will be more suitable for higher-dimensional tasks.

Verisig 2.0 can only handle NN controllers with differential activation functions (e.g., sigmoid and tanh). In the lower-dimensional benchmarks, the reachable sets computed by POLAR-Express and Verisig 2.0 almost overlap with each other. However, POLAR-Express takes much less time to compute the reachable sets in all cases. In the higher-dimensional benchmarks, compared with POLAR-Express, Verisig 2.0 either has much larger over-approximation errors (e.g., Fig 8 (g) (h) (k)) or can only compute fewer steps of reachable sets (e.g., Fig 8 (i) (j) (l)).

NNV and $\alpha, \beta$-CROWN are pure (symbolic) bound estimation techniques. As explained in Section II and can be observed in Fig 8, these methods can produce large over-approximation errors when used in reachable set computations, even if the bound estimations are relatively tight compared with function-over-approximation-based approaches. Due to the closed-loop nature of NNCSs, a large over-approximation will also slow down the reachable set computations for the subsequent control steps, as evident in the runtime of $\alpha, \beta$-CROWN + Flow* in Table III even though $\alpha, \beta$-CROWN itself is quite efficient [76].

JuliaReach's runtimes are close across many benchmarks despite the substantial differences in the system dynamics and the neural networks in those benchmarks. After breaking down those runtimes, we notice that memory allocation always constitutes the largest portion of the runtimes which makes the difference of benchmarks less significant. On the other hand, it fails to verify Benchmark 2 and Benchmark 5 due to large over-approximation errors. CORA can achieve similar runtime efficiency and over-approximation tightness as POLAR-Express in some benchmarks, despite the soundness issues as noted earlier. RINO can only handle neural networks with differential activation functions such as sigmoid and tanh. It is quite efficient on the lower-dimensional systems but is slow for the higher-dimensional systems.

## VI. Conclusion and Future Work

We present POLAR-Express, a formal reachability verification tool for NNCSs, which uses layer-by-layer propagation of TMs to compute function over-approximations of NN controllers. We provide a comprehensive comparison of POLAR-Express with existing tools and show that POLAR-Express

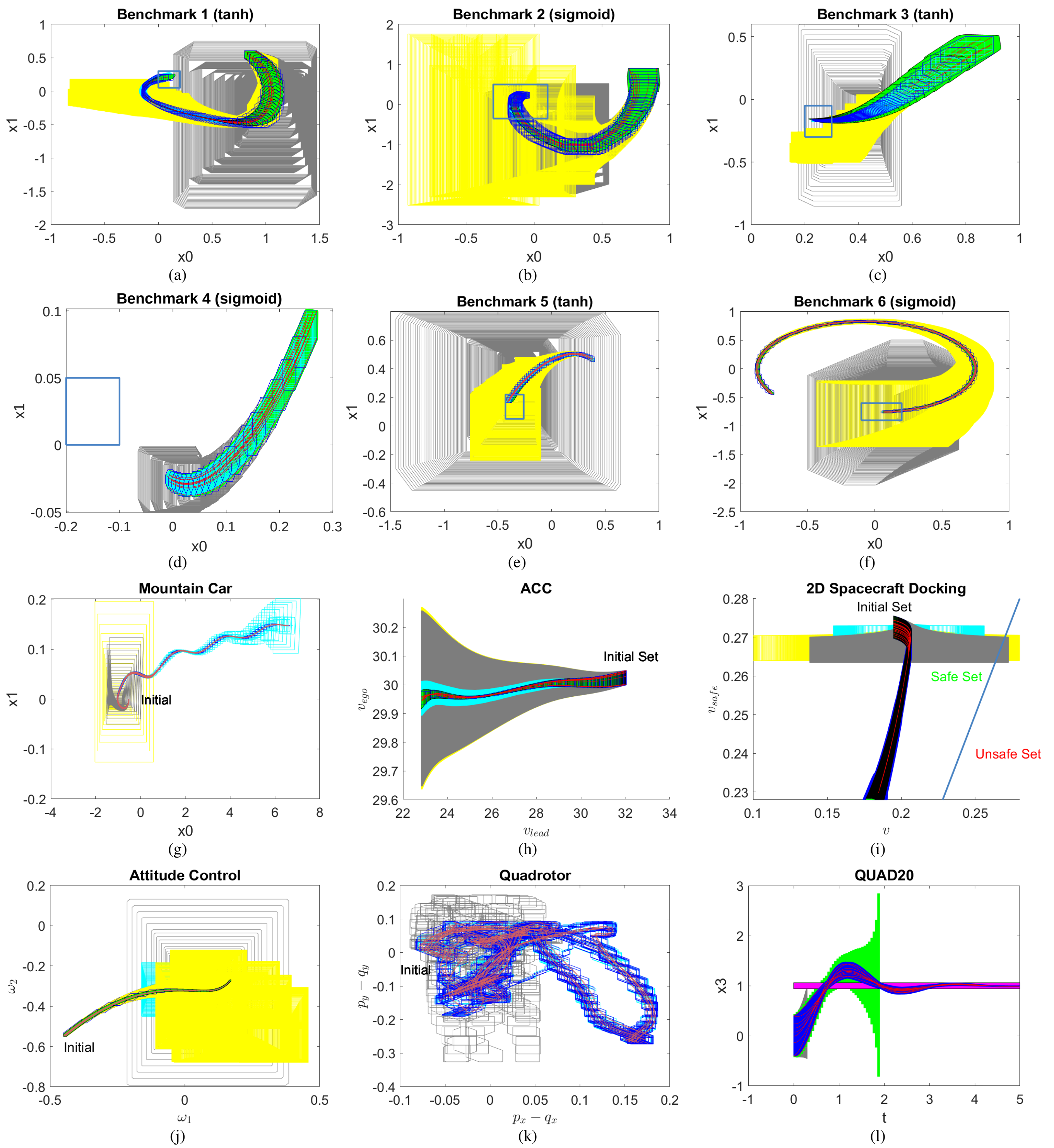



Fig. 8: Computed reachable sets of all examples by different tools. POLAR-Express (blue sets), Verisig 2.0 (cyan sets), $\alpha,\beta$-CROWN + Flow* (grey sets), NNV (yellow sets), CORA (green sets) and simulation traces (red curves), JuliaReach and RINO have their own plotting support and are not shown in this paper.

TABLE III: Verification results. *Dim* indicates the dimensions of input for NN controllers in each benchmark. NN Architecture lists activation functions and network structures in each benchmark. For example, ReLU ($n \times k$) represents that the network has $n$ hidden layers and $k$ neurons per layer. Hidden layers and the output layer use ReLU as an activation function. ReLU_tanh represents that hidden layers use ReLU and the output layer uses tanh. For each tool, we provide verification results and time in seconds. If a property cannot be verified, it is marked as U (Unknown). If a tool crashes on a benchmark with an internal error, it is marked as DNF. mt is short for multi-threads (we used 12 threads) and st is short for single-thread. The runtime of POLAR-Express is decomposed into the running time of the propagation of TMs in the neural-network controller (multi-threading support) and separately the reachability computation for the dynamical system. For neural-network controller that does not have any ReLU activation function, the single-threaded version of POLAR-Express is exactly the same as our prior prototype POLAR [46].

| Benchmarks | *Dim* | NN Architecture | POLAR-Express (mt) | POLAR-Express (st) | Verisig 2.0 (mt) | $\alpha, \beta$-CROWN + Flow* | NNV | JuliaReach | CORA | RINO |
|---|---|---|---|---|---|---|---|---|---|---|
| Benchmark 1 | 2 | ReLU (2 x 20) | Yes (0.54 + 0.08) | **Yes (0.09 + 0.08)** | – | U (230.1) | DNF | Yes (19.7) | Yes (6.7) | – |
| | | sigmoid (2 x 20) | **Yes (0.77 + 0.09)** | Yes (1.0 + 0.09) | Yes (26.6) | U (506.8) | U (14.0) | Yes (16.9) | Yes (6.7) | Yes (1.0) |
| | | tanh (2 x 20) | **Yes (1.44 + 0.18)** | Yes (2.6 + 0.14) | Yes (62.1) | U (299.3) | U (18.6) | Yes (17.2) | Yes (3.2) | Yes (1.9) |
| | | ReLU_tanh (2 x 20) | Yes (0.55 + 0.08) | **Yes (0.1 + 0.07)** | – | U (293.5) | U (18.0) | Yes (17.2) | Yes (4.6) | – |
| Benchmark 2 | 2 | ReLU (2 x 20) | Yes (0.13 + 0.02) | **Yes (0.02 + 0.01)** | – | U (112.9) | U (13.6) | U (18.8) | Yes (4.0) | – |
| | | sigmoid (2 x 20) | Yes (0.37 + 0.17) | Yes (0.6 + 0.15) | Yes (6.5) | U (134.5) | U (6.8) | U (18.9) | Yes (1.1) | **Yes (0.3)** |
| | | tanh (2 x 20) | **Yes (1.3 + 0.5)** | Yes (2.7 + 0.5) | U (4.7) | U (161.1) | U (5.9) | U (18.9) | DNF | DNF |
| | | ReLU_tanh (2 x 20) | Yes (0.26 + 0.5) | **Yes (0.1 + 0.4)** | – | U (85.5) | U (17.1) | U (18.9) | Yes (1.0) | – |
| Benchmark 3 | 2 | ReLU (2 x 20) | Yes (1.2 + 0.85) | **Yes (0.4 + 0.75)** | – | U (217.8) | U (21.0) | Yes (17.6) | Yes (4.5) | – |
| | | sigmoid (2 x 20) | Yes (3.3 + 0.8) | Yes (7.1 + 0.8) | Yes (38.2) | U (353.0) | U (26.8) | Yes (17.5) | **Yes (2.6)** | Yes (2.8) |
| | | tanh (2 x 20) | Yes (3.2 + 0.8) | Yes (6.2 + 0.7) | Yes (31.9) | U (424.6) | U (27.1) | Yes (17.7) | **Yes (2.4)** | Yes (6.5) |
| | | ReLU_sigmoid (2 x 20) | No (1.4 + 0.8) | **No (0.6 + 0.7)** | – | U (141.2) | U (20.6) | No (17.7) | No (2.2) | – |
| Benchmark 4 | 3 | ReLU (2 x 20) | Yes (0.15 + 0.02) | **Yes (0.03 + 0.02)** | – | U (75.4) | DNF | Yes (16.6) | Yes (3.0) | – |
| | | sigmoid (2 x 20) | No (0.24 + 0.02) | **No (0.17 + 0.02)** | No (5.7) | No (139.5) | DNF | No (16.7) | No (0.9) | No (0.3) |
| | | tanh (2 x 20) | No (0.23 + 0.02) | No (0.17 + 0.02) | No (4.6) | No (160.9) | DNF | No (16.9) | No (0.9) | **No (0.1)** |
| | | ReLU_tanh (2 x 20) | Yes (0.14 + 0.02) | **Yes (0.03 + 0.02)** | – | U (89.6) | U (5.4) | Yes (16.9) | Yes (1.0) | – |
| Benchmark 5 | 3 | ReLU (3 x 100) | Yes (2.4 + 0.02) | **Yes (1.8 + 0.02)** | – | U (111.4) | DNF | U (18.6) | Yes (3.4) | – |
| | | sigmoid (3 x 100) | No (4.2 + 0.02) | **No (3.7 + 0.02)** | No (93.1) | U (229.3) | DNF | U (19.0) | U (1.3) | U (7.2) |
| | | tanh (3 x 100) | Yes (4.2 + 0.02) | Yes (3.8 + 0.02) | Yes (74.2) | U (263.0) | U (125.0) | U (18.7) | **Yes (1.2)** | Yes (2.7) |
| | | ReLU_tanh (3 x 100) | Yes (2.5 + 0.02) | Yes (2.1 + 0.02) | – | U (96.1) | U (4.1) | U (18.8) | **Yes (1.7)** | – |
| Benchmark 6 | 4 | ReLU (3 x 20) | Yes (0.2 + 0.05) | **Yes (0.05 + 0.05)** | – | U (149.8) | U (8.8) | Yes (19.6) | Yes (8.6) | – |
| | | sigmoid (3 x 20) | Yes **(0.47 + 0.05)** | Yes (0.7 + 0.05) | Yes (24.3) | U (217.5) | U (9.9) | Yes (19.5) | Yes (2.7) | Yes (0.6) |
| | | tanh (3 x 20) | Yes (0.5 + 0.05) | Yes (0.7 + 0.05) | Yes (18.8) | U (259.1) | U (9.1) | Yes (20.4) | Yes (3.7) | **Yes (0.5)** |
| | | ReLU_tanh (3 x 20) | Yes (0.2 + 0.05) | **Yes (0.05 + 0.05)** | – | U (109.3) | U (5.9) | Yes (19.6) | Yes (5.1) | – |
| Discrete Mountain Car | 2 | sigmoid_tanh (2 x 16) | **Yes (3.0 + 0.12)** | Yes (4.0 + 0.14) | Yes (70.3) | U (450.0) | U (18.1) | – | – | DNF |
| ACC | 6 | tanh (3 x 20) | **Yes (2.6 + 0.15)** | Yes (4.4 + 0.14) | Yes (1980.2) | U (1806.4) | U (39.6) | U (29.9) | Yes (3.7) | Yes (10.1) |
| 2D Spacecraft Docking | 6 | tanh (2 x 64) | Yes (15.8 + 29.7) | Yes (25.5 + 29.8) | U (3331.0) | U (3119.7) | U (74.0) | Yes (37.3) | **Yes (33.5)** | DNF |
| Attitude Control | 6 | sigmoid (3 x 64) | Yes (8.0 + 0.6) | Yes (10.1 + 0.5) | U (1271.5) | U (582.6) | U (358.2) | Yes (19.9) | **Yes (1.9)** | Yes (121.3) |
| QUAD20 | 12 | sigmoid (3 x 64) | **Yes (316.8 + 212.5)** | Yes (809.4 + 197.6) | U (17939.7, 1step) | U (370.2, 3step) | U (325.5, 1step) | U (60.6, 25step) | U (2297, 20steps) | DNF |

achieves state-of-the-art efficiency and tightness in reachable set computations. On the other hand, current techniques still do not scale well to high-dimensional cases. In our experiment, the performance of Verisig 2.0 degrades significantly for the 6-dimensional examples, and POLAR-Express is also less efficient in the QUAD20 example. We believe state dimensions, control step sizes, and the number of total control steps are the key factors in scalability. As TMs are parameterized by state variables, higher state dimensions will lead to a more tedious polynomial expression in the TMs. Meanwhile, a large control step or a large number of total control steps can make it more difficult to propagate the state dependencies across the plant dynamics and across multiple control steps. We believe that addressing these scalability issues will be the main subject of future work in NNCS reachability analysis.

## References


[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[2] X. Liu, C. Huang, Y. Wang, B. Zheng, and Q. Zhu, "Physics-aware safety-assured design of hierarchical neural network based planner," in *ICCPS*. IEEE, 2022, pp. 137–146.

[3] X. Liu, R. Jiao, B. Zheng, D. Liang, and Q. Zhu, "Safety-driven interactive planning for neural network-based lane changing," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '23, 2023.

[4] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *DASC*. IEEE, 2016, pp. 1–10.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[6] S. Xu, Y. Fu, Y. Wang, Z. O'Neill, and Q. Zhu, "Learning-based framework for sensor fault-tolerant building hvac control with model-assisted learning," in *BuildSys*, 2021, pp. 1–10.

[7] T. Wei, S. Ren, and Q. Zhu, "Deep reinforcement learning for joint datacenter and hvac load control in distributed mixed-use buildings," *IEEE Transactions on Sustainable Computing*, vol. 6, no. 3, pp. 370–384, 2021.

[8] K. Julian and M. J. Kochenderfer, "Neural network guidance for UAVs," in *AIAA Guidance Navigation and Control Conference (GNC)*, 2017.

[9] Q. Zhu, W. Li, H. Kim, Y. Xiang, K. Wardega, Z. Wang, Y. Wang, H. Liang, C. Huang, J. Fan, and H. Choi, "Know the unknowns: Addressing disturbances and uncertainties in autonomous systems," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20, 2020.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[11] Y. Wang, S. S. Zhan, R. Jiao, Z. Wang, W. Jin, Z. Yang, Z. Wang, C. Huang, and Q. Zhu, "Enforcing hard constraints with soft barriers: Safe reinforcement learning in unknown stochastic environments," in *ICML*. PMLR, 2023.

[12] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *ICML*, 2004, p. 1.

[13] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," *RSS*, 2018.

[14] S. Xu, Y. Wang, Y. Wang, Z. O'Neill, and Q. Zhu, "One for many: Transfer learning for building hvac control," in *BuildSys*, 2020, pp. 230–239.

[15] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu, "Reachnn: Reachability analysis of neural-network controlled systems," *TECS*, vol. 18, no. 5s, pp. 1–22, 2019.

[16] S. Dutta, X. Chen, and S. Sankaranarayanan, "Reachability analysis for neural feedback systems using regressive polynomial rule inference," in *HSCC*. ACM Press, 2019, pp. 157–168.

[17] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: verifying safety properties of hybrid systems with neural network controllers," in *HSCC*, 2019, pp. 169–178.

[18] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "Nnv: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," in *CAV*. Springer, 2020, pp. 3–17.

[19] Y.-S. Wang, L. Weng, and L. Daniel, "Neural network control policy verification with persistent adversarial perturbation," in *ICML*. PMLR, 2020, pp. 10 050–10 059.

[20] C. Huang, J. Fan, X. Chen, W. Li, and Q. Zhu, "Divide and slide: Layer-wise refinement for output range analysis of deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3323–3335, 2020.

[21] Z. Wang, C. Huang, and Q. Zhu, "Efficient global robustness certification of neural networks via interleaving twin-network encoding," in *DATE'22: Proceedings of the Conference on Design, Automation and Test in Europe*, 2022.

[22] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, 2020.

[23] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theor. Comput. Sci.*, vol. 138, no. 1, pp. 3–34, 1995.

[24] N. S. Nedialkov, "Implementing a rigorous ode solver through literate programming," in *Modeling, Design, and Simulation of Systems with Uncertainties*, ser. Mathematical Engineering, A. Rauh and E. Auer, Eds. Springer Berlin Heidelberg, 2011, vol. 3, ch. Mathematical Engineering, pp. 3–19.

[25] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *Proc. of CAV'11*, ser. LNCS, vol. 6806. Springer, 2011, pp. 379–395.

[26] M. Althoff, "An introduction to cora 2015," in *Proc. of ARCH'15*, ser. EPiC Series in Computer Science, vol. 34, 2015, pp. 120–151.

[27] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Proc. of CAV'13*, ser. LNCS, vol. 8044. Springer, 2013, pp. 258–263.

[28] S. Kong, S. Gao, W. Chen, and E. M. Clarke, "dreach: $\delta$-reachability analysis for hybrid systems," in *Proc. of TACAS'15*, ser. LNCS, vol. 9035. Springer, 2015, pp. 200–205.

[29] D. S. Graça, J. Buescu, and M. L. Campagnolo, "Boundedness of the domain of definition is undecidable for polynomial odes," *Electronic Notes in Theoretical Computer Science*, vol. 202, pp. 49–57, 2008, proceedings of the Fourth International Conference on Computability and Complexity in Analysis (CCA 2007).

[30] T. Dreossi, T. Dang, and C. Piazza, "Parallelotope bundles for polynomial reachability," in *HSCC*. ACM, 2016, pp. 297–306.

[31] J. Lygeros, C. Tomlin, and S. Sastry, "Controllers for reachability specifications for hybrid systems," *Automatica*, vol. 35, no. 3, pp. 349–370, 1999.

[32] Z. Yang, C. Huang, X. Chen, W. Lin, and Z. Liu, "A linear programming relaxation based approach for generating barrier certificates of hybrid systems," in *FM*. Springer, 2016, pp. 721–738.

[33] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *HSCC*. Springer, 2004, pp. 477–492.

[34] C. Huang, X. Chen, W. Lin, Z. Yang, and X. Li, "Probabilistic safety verification of stochastic hybrid systems using barrier certificates," *TECS*, vol. 16, no. 5s, p. 186, 2017.

[35] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *CAV*. Springer, 2017, pp. 3–29.

[36] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *CAV*. Springer, 2017, pp. 97–117.

[37] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Learning and verification of feedback control systems using feedforward neural networks," *IFAC-PapersOnLine*, vol. 51, no. 16, pp. 151–156, 2018.

[38] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1599–1614.

[39] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *NeurIPS*, 2018, pp. 10 802–10 813.

[40] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification," *NeurIPS*, vol. 34, 2021.

[41] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*. SIAM, 2009.

[42] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," *NeurIPS*, vol. 31, 2018.

[43] J. Fan, C. Huang, X. Chen, W. Li, and Q. Zhu, "Reachnn*: A tool for reachability analysis of neural-network controlled systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2020, pp. 537–542.

[44] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. Pappas, and I. Lee, "Verisig 2.0: Verification of neural network controllers using taylor model preconditioning," in *CAV*, A. Silva and K. R. M. Leino, Eds. Cham: Springer International Publishing, 2021, pp. 249–262.

[45] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verifying the safety of autonomous systems with neural network controllers," *TECS*, vol. 20, no. 1, pp. 1–26, 2020.

[46] C. Huang, J. Fan, X. Chen, W. Li, and Q. Zhu, "Polar: A polynomial arithmetic framework for verifying neural-network controlled systems," in *Automated Technology for Verification and Analysis: 20th International Symposium, ATVA 2022, Virtual Event, October 25–28, 2022, Proceedings*. Springer, 2022, pp. 414–430.

[47] E. Goubault and S. Putot, "RINO: robust inner and outer approximated reachability of neural networks controlled systems," in *Proc. of CAV'22*, ser. LNCS, S. Shoham and Y. Vizel, Eds., vol. 13371. Springer, 2022, pp. 511–523.

[48] N. Kochdumper, H. Krasowski, X. Wang, S. Bak, and M. Althoff, "Provably safe reinforcement learning via action projection using reachability analysis and polynomial zonotopes," *CoRR*, vol. abs/2210.10691, 2022.

[49] C. Schilling, M. Forets, and S. Guadalupe, "Verification of neural-network control systems by integrating taylor models and zonotopes," in *Proc. of AAAI'22*. AAAI Press, 2022, pp. 8169–8177.

[50] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," *IEEE Transactions on Automatic Control*, vol. 67, no. 1, pp. 1–15, 2022.

[51] C. Huang, S. Xu, Z. Wang, S. Lan, W. Li, and Q. Zhu, "Opportunistic intermittent control with safety guarantees for autonomous systems," *Design Automation Conference (DAC'20)*, 2020.

[52] Y. Wang, C. Huang, and Q. Zhu, "Energy-efficient control adaptation with safety guarantees for learning-enabled cyber-physical systems," in *ICCAD*, 2020, pp. 1–9.

[53] Z. Wang, C. Huang, H. Kim, W. Li, and Q. Zhu, "Cross-layer adaptation with safety-assured proactive task job skipping," vol. 20, no. 5s, sep 2021. [Online]. Available: https://doi.org/10.1145/3477031

[54] X. Chen and S. Sankaranarayanan, "Reachability analysis for cyber-physical systems: Are we there yet?" in *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings*, ser. Lecture Notes in Computer Science, vol. 13260. Springer, 2022, pp. 109–130.

[55] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis*. Springer, 2001.

[56] G. M. Ziegler, *Lectures on Polytopes*, ser. Graduate Texts in Mathematics. Springer, 1995, vol. 152.

[57] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NASA Formal Methods Symposium*. Springer, 2018, pp. 121–138.

[58] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *ICLR*, 2019.

[59] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2017, Conference Proceedings, pp. 251–268.

[60] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward relu neural networks," *CoRR*, vol. abs/1706.07351, 2017. [Online]. Available: http://arxiv.org/abs/1706.07351

[61] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," in *IJCAI*, 2018, pp. 2651–2659.

[62] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 3–18.

[63] M. Berz and K. Makino, "Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models," *Reliable Computing*, vol. 4, pp. 361–369, 1998.

[64] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Taylor model flowpipe construction for non-linear hybrid systems," in *Proc. of RTSS'12*. IEEE Computer Society, 2012, pp. 183–192.

[65] X. Chen, "Reachability analysis of non-linear hybrid systems using taylor models," Ph.D. dissertation, RWTH Aachen University, 2015.

[66] X. Chen and S. Sankaranarayanan, "Decomposed reachability analysis for nonlinear systems," in *RTSS*. IEEE Press, Nov 2016, pp. 13–24.

[67] M. Althoff, "Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets," in *Proc. of HSCC 2013*. ACM, 2013, p. 173–182.

[68] L. Perko, *Differential Equations and Dynamical Systems (3rd edition)*. Springer, 2006.

[69] U. Ravaioli, J. Cunningham, J. McCarroll, V. Gangal, K. Dunlap, and K. Hobbs, "Safe reinforcement learning benchmark environments for aerospace control systems," in *2022 IEEE Aerospace Conference*. IEEE, 2022.

[70] V. D. Blondel and J. N. Tsitsiklis, "Overview of complexity and decidability results for three classes of elementary nonlinear systems," in *Learning, control and hybrid systems*. Springer London, 1999, pp. 46–58.

[71] K. Makino and M. Berz, "Taylor models and other validated functional inclusion methods," *J. Pure and Applied Mathematics*, vol. 4, no. 4, pp. 379–456, 2003.

[72] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel, "Evaluating the robustness of neural networks: An extreme value theory approach," *arXiv preprint arXiv:1801.10578*, 2018.

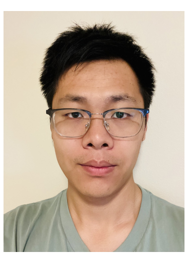
[73] G. G. Lorentz, *Bernstein Polynomials*. American Mathematical Society, 2013.

[74] T. N. T. Goodman, H. Oruç, and G. M. Phillips, "Convexity and generalized bernstein polynomials," *Proceedings of the Edinburgh Mathematical Society*, vol. 42, no. 1, p. 179–190, 1999.

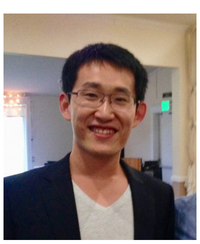
[75] D. M. Lopez, M. Althoff, L. Benet, X. Chen, J. Fan, M. Forets, C. Huang, T. T. Johnson, T. Ladner, W. Li *et al.*, "Arch-comp22 category report: artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants," in *Proceedings of 9th International Workshop on ARCH22*, 2022.

[76] M. N. Müller, C. Brix, S. Bak, C. Liu, and T. T. Johnson, "The third international verification of neural networks competition (vnn-comp 2022): summary and results," *arXiv preprint arXiv:2212.10376*, 2022.


## VII. Biography

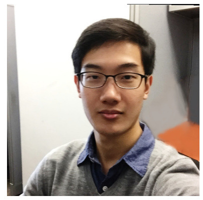


**Yixuan Wang** (wangyixu14@gmail.com) is a doctoral student at the ECE Department, Northwestern University, Evanston, IL, USA. His research focuses on design, adaptation, and verification for learning-enabled cyber-physical systems by exploring the intersection of formal methods and machine learning.



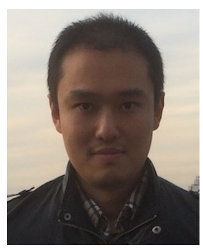
**Weichao Zhou** (zwc662@bu.edu) is a doctoral student at the Department of Electrical and Computer Engineering, Boston University, Boston, MA, USA. His research focuses on reinforcement learning, formal verification for learning-enabled control systems, and specification-guided imitation learning.

**Jiameng Fan** (jmfan@bu.edu) received the Ph.D. degree in Electrical Engineering from Boston University, Boston, MA, USA. He is working as a software engineer at Google LLC.



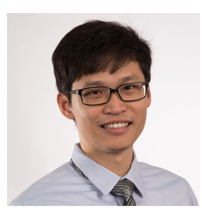
**Zhilu Wang** (zhilu.wang@u.northwestern.edu) received his Ph.D. degree in 2022, from the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL, USA. He is working as a software engineer at Google LLC.

**Jiajun Li** (j.li234@liverpool.ac.uk) is a doctoral student at the Department of Computer Science, University of Liverpool, Liverpool, UK. His research interests include formal methods, program synthesis, and high-performance computing.



**Xin Chen** (chenxin@unm.edu) received his Doctor rerum naturalium (Doctor of natural sciences) from RWTH Aachen University, Germany in 2015. He is currently an assistant professor of Computer Science at the University of New Mexico, Albuquerque, NM, USA. His research interests mainly focus on solving the safety and security problems for the dynamical systems equipped with AI controllers using numerical and formal methods.

**Chao Huang** (chao.huang2@liverpool.ac.uk) is a Lecturer (Assistant Professor) at the Department of Computer Science, University of Liverpool, Liverpool, UK. He is also an adjunct assistant professor in the Department of Electrical and Computer Engineering at Northwestern University, US. His research interests include design and verification of intelligent systems, of which the components involve machine learning techniques.



**Wenchao Li** (wenchao@bu.edu) is an Assistant Professor at the Department of Electrical and Computer Engineering at Boston University, Boston, MA, USA. He received his Ph.D. in Electrical Engineering and Computer Sciences from the University of California, Berkeley. His research sits at the intersection of formal methods and machine learning, with a focus on building safe and trustworthy autonomous systems.

**Qi Zhu** (qzhu@northwestern.edu) is an Associate Professor at the Department of Electrical and Computer Engineering in Northwestern University, Evanston, IL, USA. He received his Ph.D. in Electrical Engineering and Computer Sciences from the University of California, Berkeley. His research interests include design automation for cyber-physical systems (CPS) and Internet-of-Things (IoT), safe and robust machine learning for CPS and IoT, cyber-physical security, and system-on-chip design.