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

Frank Yang
frankyang2024@u.northwestern.edu
Northwestern University
Evanston, Illinois, USA

Sinong Simon Zhan
Northwestern University
Evanston, Illinois, USA
sinongzhan2028@u.northwestern.edu

Yixuan Wang
Northwestern University
Evanston, Illinois, USA
yixuanwang2024@u.northwestern.edu

Chao Huang
University of Southampton
Southampton, UK
chao.Huang@soton.ac.uk

Qi Zhu
Northwestern University
Evanston, Illinois, USA
qzhu@u.northwestern.edu

1 Introduction

Neural networks (NNs) playing the role of controllers have demonstrated impressive empirical performances on challenging control problems. However, the potential adoption of NN controllers in real-life applications also raises a growing concern over the safety of these neural network-controlled systems (NNCSs), especially when used in safety-critical applications. We present POLAR-express [12], a reachability analysis tool for safety verification of NNCSs based on polynomial arithmetic. POLAR-Express uses a novel approach to iteratively over-approximate the neuron output ranges layer-by-layer with a combination of Bernstein polynomial approximation for continuous activation functions and Taylor Model (TM) arithmetic. This approach can overcome the main drawback in standard TM's inability to handle functions that cannot be well approximated by Taylor polynomials, and significantly improve the accuracy and efficiency of reachable states computation for NNCSs. POLAR-Express offers a symbolic remainder approach when estimating the output range of a neural network, to tighten the over-approximation. The tool supports general feed-forward neural networks with continuous but not necessarily differentiable activation functions. POLAR-Express also features multi-threading support to parallelize the computation for layer-by-layer propagation to speed up the verification process.

To the best of our knowledge, POLAR-Express is the state-of-the-art verification tool for the reachability analysis of NNCSs. Compared to other existing tools [2, 5, 7, 9–11], it generates the most accurate and tightest reachable set with the highest efficiency. We hope POLAR-Express can serve as an accessible introduction to these analysis techniques for those who wish to apply them to verify NNCSs in their own applications. We also provide installation instructions and example usage of POLAR-Express in this work. The source code is available at https://github.com/ChaoHuang2018/POLAR_Tool

2 POLAR-Express

2.1 Background

NNCS. NNCSs are a class of cyber-physical systems where a neural network serves as the primary controller. As shown in Fig. 1, we consider the **plant dynamics** of an NNCS as $\dot{x} = f(x, u)$ where the state variable is $x \in \mathcal{X} \subseteq \mathbb{R}^n$, control input is $u \in \mathcal{U} \subseteq \mathbb{R}^m$,

ESSC, September 29th – October 4th, 2024, 2018.

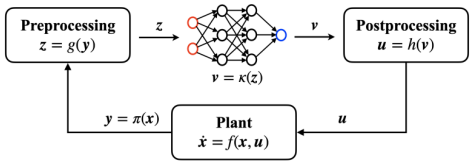


Figure 1: Formal Definition of NNCS

and the dynamic $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is a Lipschitz continuous function, ensuring a unique solution of the ODE. Such a system can be controlled by a feedback NN controller $\kappa(\cdot)$ at i-th ($i = 0, 1, \cdots$) sampling period $i\delta$. In the **preprocessing** phase, $\kappa(\cdot)$ retrieves sensor measurements y and transforms them to the NN controller's input format $g(y)$. In the **postprocessing** phase, the NNCS system reads system state $x_{i\delta}$, generates a control input $u = h(v)$ from the NN controller output $v = \kappa(x_{i\delta})$, and the system evolves according to $\dot{x} = f(x, u)$ within the period of $[i\delta, (i+1)\delta]$. The *flowmap* function $\varphi(x_0, t) : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ describes the solution of the NNCS, which maps the initial state x_0 to the system state $\varphi(x_0, t)$ at time t starting from x_0. We call a state x' *reachable* if there exist $x_0 \in \mathcal{X}$ and $t \in \mathbb{R}_{\geq 0}$ with $x' = \varphi(x_0, t)$. A reachable set $\mathcal{X}r^T$ is a collection of all reachable states within a time range $T = \mathbb{R}_{\geq 0}$ given an initial space $\mathcal{X}_0 = x_0$, i.e., $\mathcal{X}_r^T = \{\varphi(x_0, t) | x_0 \in \mathcal{X} \wedge t \in T\}$. Intuitively, once the reachable set $\mathcal{X}_r^T$ is non-overlapping with the unsafe sets $\mathcal{X}_u$, safety is guaranteed for such an NNCS throughout the time horizon T.

POLAR-Express. POLAR-Express is a state-of-the-art tool developed to compute an over-approximation of the reachable set for NNCSs. This over-approximation is crucial for safety verification, as it provides a guaranteed upper bound on all possible system behaviors. If the over-approximated reachable set does not intersect with any unsafe states, then the system is provably safe.

2.2 Techniques of POLAR-Express

POLAR-Express computes the reachable set for NNCSs using the framework outlined in Algorithm 1. It iteratively computes flowpipes that over-approximate the reachable states of the system over discrete time steps. The most critical aspect of this process is the over-approximation of the neural network controller's behavior, which is achieved through the layer-by-layer propagation method detailed in Algorithm 2. This section addresses each formulation in detail.

Algorithm 1 Framework of POLAR-Express

Require: Plant dynamics $x' = f(x,u)$, preprocessing $g(\cdot)$, post-processing $h(\cdot)$, NN controller $\kappa(\cdot)$, number of control steps $\mathcal{N}$, initial set X_0

Ensure: Over-approximation of the reachable set over the time interval of $[0, K\delta]$ where δ_c is the control step size

$\mathcal{R} \leftarrow \emptyset$
$\mathcal{X}_0 \leftarrow X_0$
for $i = 0$ to $\mathcal{N} - 1$ **do**
    Computing a superset $\mathcal{Y}_i$ for the range of $\pi(\mathcal{X}_i)$
    Computing a superset $\mathcal{Z}_i$ for the range of $g(\mathcal{Y}_i)$
    Computing a superset $\mathcal{V}_i$ for the range of $\kappa(\mathcal{Z}_i)$, with multi-threading support
    Computing a superset $\mathcal{U}_i$ for the range of $h(\mathcal{V}_i)$
    Computing a set $\mathcal{F}$ of flowpipes for the continuous dynamics $\dot{x} = f(x,u)$ with $u \in \mathcal{U}_i$ from the initial set $x(0) \in \mathcal{X}_i$ over the time interval of $[i\delta, (i+1)\delta]$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{F}$
    Evaluating an overapproximation for the reachable set at the time $t = (i+1)\delta$ based on $\mathcal{F}$ and assigning it to $\mathcal{X}_{i+1}$
end for
return $\mathcal{R}$

Layer-by-layer Propagation with Multi-Threading Support. POLAR-Express uses the layer-by-layer propagation scheme to compute a TM output for the NN. Algorithm 2 presents computing tighter TM over-approximation for the activation functions. First, we compute the polynomial approximations for the activation functions of the neurons in the current layer. Second, interval remainders are evaluated for those polynomials to ensure that for each TM of the activation function. Third, $p_{i+1}(x_0, I_{i+1})$ is computed as the TM composition. 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. The computation of those TMs can be conducted in parallel, retaining time when the dimension of NN layers is large.

More precise Overapproximation fo ReLU functions. POLAR-Express achieves a more precise and efficient Bernstein overapproximation for ReLU activation functions by leveraging the convexity property of ReLU. For a ReLU function over a domain $[a,b]$ that includes 0, the method computes an order-k Bernstein polynomial $p(x)$ as an upper bound. Then, it creates a tight lower bound by shifting $p(x)$ down by $p(0)/2$. The resulting Taylor model $(p(x) - 0.5\varepsilon, [-0.5\varepsilon, 0.5\varepsilon])$, where $\varepsilon = p(0)$, provides a guaranteed overapproximation of the ReLU function. This approach is both more precise, as it tightly bounds the ReLU function from above and below, and more efficient, as it avoids the need for additional sampling or complex remainder calculations typically required for other activation functions.

Symblic Remainder. To tighten the over-approximation, POLAR-Express stores the TM intervals symbolically with their linear transformation matrix and only evaluates the remainder interval at the end. This approach is called symbolic remainder, which reduces the accumulation of over-approximation error in TM by avoiding the wrapping effect in linear mappings.

Algorithm 2 Layer-by-layer propagation using polynomial arithmetic and TMs

Require:
Input TM $(p_1(x_0), I_1)$ with $x_0 \in X_0$
Weight matrices $W_1, \ldots, W_{M+1}$ for hidden and output layers
Bias vectors $B_1, \ldots, B_{M+1}$ for hidden and output layers

Ensure: TM $(p_r(x_0), I_r)$ containing the set $\kappa((p_1(x_0), I_1))$

$(p_r, I_r) \leftarrow (p_1, I_1)$
for $i = 1$ to $M + 1$ **do**
    $(p_t, I_t) \leftarrow W_i \cdot (p_r, I_r) + B_i$
    Compute polynomial approximation $p_{\sigma,i}$ for activation functions σ w.r.t. domain (p_t, I_t)
    Evaluate conservative remainder $I_{\sigma,i}$ for $p_{\sigma,i}$ w.r.t. domain (p_t, I_t)
    $(p_r, I_r) \leftarrow p_{\sigma,i}(p_t + I_t) + I_{\sigma,i}$
end for
return (p_r, I_r)

3 Benchmark Evaluations

Benchmarks. Our NNCS benchmark suite encompasses a diverse set of control tasks with varying complexity, derived from multiple sources [1, 3, 4, 6, 8, 10]. The suite includes:

- *Benchmarks 1-6*: These involve 2-4D ODEs with 10-60 verification steps to verify if NNCS can reach the target set from any initial state in $\mathcal{N}$ control steps.
- *Discrete Mountain Car*: This 2D NNCS describes an underpowered car driving up a steep hill with 150 control steps.
- *Adaptive Cruise Control*: This 6D system models vehicles over 50 steps to maintain a safe distance between the vehicles.
- *2D Spacecraft Docking*: This 5D system has 120 control steps and focuses on maintaining a safe velocity throughout the docking process.
- *Attitude Control*: This system models 6D spacecraft attitude dynamics.
- *Quadrotor-MPC*: This 9D hybrid system features a quadrotor with constant velocity over 30 control steps.
- *QUAD20*: This system models 12D quadrotor dynamics.

These benchmarks are designed to test scalability across system dimensionality, time horizons, and neural network architectures. The neural network controllers employed in these systems range from 2-3 hidden layers with 16-100 neurons per layer, utilizing activation functions such as ReLU, sigmoid, and tanh. The primary objective of these benchmarks is to evaluate reachability. Specifically, each task verifies if the NNCS can reach a defined target set from any initial state within $\mathcal{N}$ control steps. Tool performance is assessed based on its ability to prove these reachability tasks. For a comprehensive understanding of these benchmarks, including detailed specifications and implementations, we refer readers to the POLAR-express GitHub repository at https://github.com/ChaoHuang2018/POLAR_Tool/tree/main/POLAR_Express_Benchmarks.

Benchmarks	Dim	NN Architecture	POLAR-Express (mt)	POLAR-Express (st)
Benchmark 1	2	ReLU (2 x 20)	Yes (0.54 ± 0.08)	Yes (0.09 ± 0.08)
		sigmoid (2 x 20)	Yes (0.77 ± 0.09)	Yes (1.0 ± 0.09)
		tanh (2 x 20)	Yes (1.44 ± 0.18)	Yes (2.6 ± 0.14)
		ReLU_tanh (2 x 20)	Yes (0.55 ± 0.08)	Yes (0.1 ± 0.07)
Benchmark 2	2	ReLU (2 x 20)	Yes (0.13 ± 0.02)	Yes (0.02 ± 0.01)
		sigmoid (2 x 20)	Yes (0.37 ± 0.17)	Yes (0.6 ± 0.15)
		tanh (2 x 20)	Yes (1.3 ± 0.5)	Yes (2.7 ± 0.5)
		ReLU_tanh (2 x 20)	Yes (0.26 ± 0.5)	Yes (0.1 ± 0.4)
Benchmark 3	2	ReLU (2 x 20)	Yes (1.2 ± 0.85)	Yes (0.4 ± 0.75)
		sigmoid (2 x 20)	Yes (3.3 ± 0.8)	Yes (7.1 ± 0.8)
		tanh (2 x 20)	Yes (3.2 ± 0.8)	Yes (6.2 ± 0.7)
		ReLU_sigmoid (2 x 20)	No (1.4 ± 0.8)	No (0.6 ± 0.7)
Benchmark 4	3	ReLU (2 x 20)	Yes (0.15 ± 0.02)	Yes (0.03 ± 0.02)
		sigmoid (2 x 20)	No (0.24 ± 0.02)	No (0.17 ± 0.02)
		tanh (2 x 20)	No (0.23 ± 0.02)	No (0.17 ± 0.02)
		ReLU_tanh (2 x 20)	Yes (0.14 ± 0.02)	Yes (0.03 ± 0.02)
Benchmark 5	3	ReLU (3 x 100)	Yes (2.4 ± 0.02)	Yes (1.8 ± 0.02)
		sigmoid (3 x 100)	No (4.2 ± 0.02)	No (3.7 ± 0.02)
		tanh (3 x 100)	Yes (4.2 ± 0.02)	Yes (3.8 ± 0.02)
		ReLU_tanh (3 x 100)	Yes (2.5 ± 0.02)	Yes (2.1 ± 0.02)
Benchmark 6	4	ReLU (3 x 20)	Yes (0.2 ± 0.05)	Yes (0.05 ± 0.05)
		sigmoid (3 x 20)	Yes (0.47 ± 0.05)	Yes (0.7 ± 0.05)
		tanh (3 x 20)	Yes (0.5 ± 0.05)	Yes (0.7 ± 0.05)
		ReLU_tanh (3 x 20)	Yes (0.2 ± 0.05)	Yes (0.05 ± 0.05)
Discrete Mountain Car	2	sigmoid_tanh (2 x 16)	Yes (3.0 ± 0.12)	Yes (4.0 ± 0.14)
ACC	6	tanh (3 x 20)	Yes (2.6 ± 0.15)	Yes (4.4 ± 0.14)
2D Spacecraft Docking	6	tanh (2 x 64)	Yes (15.8 ± 29.7)	Yes (25.5 ± 29.8)
Attitude Control	6	sigmoid (3 x 64)	Yes (8.0 ± 0.6)	Yes (10.1 ± 0.5)
QUAD20	12	sigmoid (3 x 64)	Yes (316.8 ± 212.5)	Yes (809.4 ± 197.6)

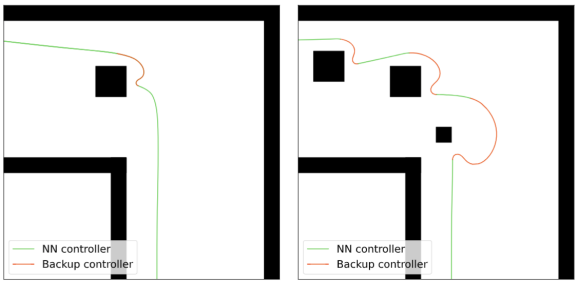
Figure 2: Verification results. *Dim* indicates the dimensions of input for NN controllers in each benchmark. *NN Architecture* lists activation functions and network structures in each benchmark. For each tool, we provide verification results and time in seconds. mt is short for multi-threads (12 threads) and st is short for single-thread

Stopping Criteria. Every test produces one of the following four results: **(Yes)** the reachability property is proved, **(No)** the reachability property is disproved, **(U)** the computed over-approximation is too large to prove or disprove the property with the best tool setting we can find, **(DNF)** a tool or system error is reported and the reachability computation fails.

Benchmark Results. According to the experimental results in Fig. 2, POLAR-Express demonstrates state-of-the-art performance across all benchmarks. It successfully verifies each task, achieving the tightest reachable set computations and the highest runtime efficiency. The multi-threading (mt) support greatly benefits higher-dimensional systems, reducing runtimes compared to the single-threading (st) version, especially evident in complex benchmarks like QUAD20 and 2D Spacecraft Docking.

A Recent Runtime Verification Benchmark. We performed a runtime case study for a Turtlebot navigation system equipped with LiDAR [13]. The Turtlebot, controlled by a neural network (NN) for steering, navigates a left turn scenario with obstacles. POLAR-Express is employed to compute the reachable sets of the NN controller κ_{nn} in real-time, projecting the Turtlebot's potential positions and orientations **30 steps** into the future. If POLAR-Express detects that the reachable set intersects with any unsafe state, the system switches control to a backup obstacle avoidance controller κ_b. Once κ_b has steered the Turtlebot out of danger, POLAR-Express continues to evaluate the safety of κ_{nn}. When the computed reachable sets for κ_{nn} no longer intersect with obstacles, control is transferred back to the NN controller.

POLAR-express operates effectively in both single (3a) and multiple obstacle avoidance scenarios (3b). In the multiple-obstacles scenario, our runtime framework consistently manages several controller switches, ensuring safety throughout the operation (3c). This



(a) Single Obstacle (b) Multi-Obstacles

Figure 3: The navigation trajectory of Turtlebot with our runtime verification based control by κ_{nn} (green) and κ_b (red) for a). navigating around a single obstacle, and b). navigating through multiple obstacles.

controller switching strategy, guided by POLAR-Express's real-time safety verification, ensures that the Turtlebot maintains safe navigation while maximizing using the more sophisticated NN controller when conditions allow.

4 How to use POLAR-Express

4.1 Installation

System Requirements: Ubuntu 18.04, MATLAB 2016a or later. POLAR relies on the Taylor model arithmetic library provided by Flow*. Please install Flow* with the same directory as POLAR. You can either use the following command or follow the manual of Flow* for installation.

Install dependencies through apt-get install

```
sudo apt-get install m4 libgmp3-dev libmpfr-dev
libmpfr-doc libgsl-dev gsl-bin bison flex gnuplot-x11
libglpk-dev gcc-8 g++-8 libopenmpi-dev
libpthread-stubs0-dev
```

Download Flow*

```
git clone https://github.com/chenxin415/flowstar.git
```

Compile Flow*

```
cd flowstar/flowstar-toolbox
make
```

4.2 Example Usage

Consider the following nonlinear system (see Benchmark 1 in the source code)

$$\begin{bmatrix} \dot{x_0} \\ \dot{x_1} \\ \dot{u} \end{bmatrix} = \begin{bmatrix} x_1 \\ ux_1^2 - x_0 \\ 0 \end{bmatrix}$$

We declare the state and input variables (see reachnn_benchmark_1.cpp)

```
// Declaration of the state and input variables.
unsigned int numVars = 4;
Variables vars;

int x0_id = vars.declareVar("x0");
int x1_id = vars.declareVar("x1");
```

```
// declaration of time variable t
int t_id = vars.declareVar("t");
int u_id = vars.declareVar("u");

// Define the continuous dynamics.
ODE<Real> dynamics({"x1", "u*x1^2-x0",
                    "1", "0"}, vars);
```

Note that "1" and "0" in dynamics means $\dot{t} = 1$, and $\dot{u} = 0$, respectively. Consider an initial set x_0, x_1, u of (0.85, 0.55, 0) respectively, using a NN controller with ReLU_tanh activation function,

```
// Define initial state set
Interval init_x0(0.85 - w, 0.85 + w),
         init_x1(0.55 - w, 0.55 + w),
         init_u(0);

// Define the neural network controller.
NeuralNetwork nn(nn_1_relu_tanh);
```

where w is the width of the initial set. To define the target set,

```
// Define target set
vector<Constraint> targetSet;
Constraint c1("x0 - 0.2", vars);   // x0 <= 0.2
Constraint c2("-x0", vars);        // x0 >= 0
Constraint c3("x1 - 0.3", vars);   // x1 <= 0.3
Constraint c4("-x1 + 0.05", vars); // x1 >= 0.05
```

The NN controller is specified in nn_1_relu_tanh file as follows

```
2 // number of inputs
1 // number of outputs
2 // number of hidden layers
20 // number of nodes in the first hidden layer
20 // number of nodes in the second hidden layer
ReLU // Activation function of the first layer
ReLU // Activation function of the second layer
tanh // Activation function of the output layer
// Weights of the first hidden layer
-0.0073867239989340305
...
// Bias of the first hidden layer
-0.07480818033218384
...
0 // Offset of the neural network
4 // Scala of the neural network
```

Then we can verify the NNCS with the following command under

```
make reachnn_benchmark_1 && ./reachnn_benchmark_1
0.05 35 4 6 1 relu_tanh
```

where 0.05 is the width of the initial set, 35 is the total steps that need to be verified, 4 is the order of Bernstein Polynomial, 6 is the order of Taylor Model, 1 specifies option to use symbolic remainder and relu_tanh specifies the NN controller with ReLU and tanh activation functions which points to nn_1_relu_tanh file. Fig. 4 shows the computed flowpipe from the above command.
The output file of POLAR shows the verification results

```
// Verification result at the 35th step
Verification result: Yes(35)
// Total computation time in seconds
Running Time: 11.000000 seconds
```

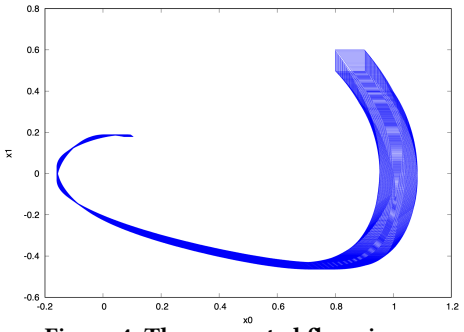


Figure 4: The computed flowpipes

5 Future Directions

While POLAR-Express represents a significant advance in NNCS verification, scalability to high-dimensional systems remains a challenge. Future work may focus on addressing this limitation, potentially through more advanced decomposition techniques or by leveraging structure in specific classes of neural network controllers.

References

[1] S. Dutta, X. Chen, and S. Sankaranarayanan. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Hybrid Systems: Computation and Control (HSCC)*. ACM Press, 157–168.

[2] E. Goubault and S. Putot. 2022. RINO: Robust inner and outer approximated reachability of neural networks controlled systems. In *Proceedings of CAV 2022 (Lecture Notes in Computer Science, Vol. 13371)*, S. Shoham and Y. Vizel (Eds.). Springer, 511–523.

[3] C. Huang, J. Fan, X. Chen, W. Li, and Q. Zhu. 2022. POLAR: A polynomial arithmetic framework for verifying neural-network controlled systems. In *Automated Technology for Verification and Analysis: 20th International Symposium, ATVA 2022*. Springer, Virtual Event, 414–430.

[4] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu. 2019. ReachNN: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–22.

[5] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. Pappas, and I. Lee. 2021. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *Computer Aided Verification*, A. Silva and K. R. M. Leino (Eds.). Springer International Publishing, Cham, 249–262.

[6] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. 2019. 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*. 169–178.

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

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

[9] C. Schilling, M. Forets, and S. Guadalupe. 2022. Verification of neural-network control systems by integrating taylor models and zonotopes. In *Proceedings of AAAI 2022*. AAAI Press, 8169–8177.

[10] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson. 2020. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*. Springer, 3–17.

[11] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter. 2021. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems* 34 (2021).

[12] Y. Wang, W. Zhou, J. Fan, Z. Wang, J. Li, X. Chen, C. Huang, W. Li, and Q. Zhu. 2023. POLAR-Express: Efficient and Precise Formal Reachability Analysis of Neural-Network Controlled Systems. In *arXiv preprint*. https://arxiv.org/abs/2304.01218

[13] Frank Yang, Sinong Simon Zhan, Yixuan Wang, Chao Huang, and Qi Zhu. 2024. Case Study: Runtime Safety Verification of Neural Network Controlled System. In *Runtime Verification*.