Approximate Conformance Checking for Closed-Loop Systems With Neural Network Controllers

P. Habeeb, Lipsy Gupta, and Pavithra Prabhakar

***Abstract*—In this article, we consider the problem of checking approximate conformance of closed-loop systems with the same plant but different neural network (NN) controllers. First, we introduce a notion of approximate conformance on NNs, which allows us to quantify semantically the deviations in closed-loop system behaviors with different NN controllers. Next, we consider the problem of computationally checking this notion of approximate conformance on two NNs. We reduce this problem to that of reachability analysis on a combined NN, thereby, enabling the use of existing NN verification tools for conformance checking. Our experimental results on an autonomous rocket landing system demonstrate the feasibility of checking approximate conformance on different NNs trained for the same dynamics, as well as the practical semantic closeness exhibited by the corresponding closed-loop systems.**

***Index Terms*—Closed-loop systems, conformance checking, neural network (NN) controller, reachability analysis.**

I. Introduction

NEURAL networks are being increasingly deployed in safety critical applications for control, perception and decision making. On one hand, they enable the handling of uncertainty and dynamism in the environment through retraining as more and more data becomes available. On the other hand, this adds to the complexity of verification and their certification. One potential way to handle the evolving nature of neural network (NN) controllers is to provide a mechanism to transfer safety proofs established with one version to another. More precisely, consider a closed-loop system (D, N) where D is the system dynamics and N is a NN controller. Let us say that we have established the safety of (D, N), that is, the reachable states R of (D, N) do not intersect the unsafe set of states U. As more data becomes available N evolves into N', and our objective is to establish that the closed-loop system is still correct. Instead of starting the verification from scratch, we want to "reuse" or transfer the safety proof of (D, N) to that of (D, N'). One approach to tackle this would be to establish closeness of N' to N, and exploit that to establish the closeness of (the reachable sets of) (D, N) and (D, N') and use that to establish the safety of the two systems. For instance, if we can conclude that the Hausdorff distance between the reachable sets of (D, N) and (D, N') are within ζ, then we only need to check if the already computed reachable set of (D, N), namely, R is at a distance of at least ζ from the unsafe set U. In this article, we consider a fundamental problem toward achieving this goal by investigating the questions of proximity of NNs and how this affects the reachable sets of closed-loop systems in which they are employed.

First, let us consider the problem of closeness of two NNs. The notion of ϵ-closeness between NNs has been explored in the literature, wherein two NNs N_1 and N_2 are said to be ϵ-close, if on the same input, their outputs are at most ϵ apart. Several methods to check ϵ-closeness have been explored, including ReluDiff [1], that proposes a symbolic interval analysis technique, StarDiff [2] that explores a geometric path enumeration-based technique, and an SMT-based approach [3]. All these works focus on NNs in isolation, while we concentrate on checking conformance between two closed-loop systems.

It has been observed [4] that the notion of ϵ-closeness is not conducive to providing a bound on the semantic closeness of closed-loop systems. Consider two closed-loop systems each with the same dynamics D and two different NNs N_1 and N_2 that are ϵ-close. Our objective is to bound the distance between the states of the systems (D, N_1) and (D, N_2) after a certain number of iterations assuming we start from the same state. While the same input is fed to both NNs in the first iteration, due to ϵ-closeness and resulting deviation in their outputs, the inputs in the next iteration are not the same. Hence, we need a notion of closeness that can account for the deviation in inputs. We propose the notion of (L, ϵ)-conformance between two NNs that stipulates that the outputs differ by at most $L\delta + \epsilon$ when the inputs differ by at most δ.

Next, we wish to establish the distance between the reachable sets of the closed-loop systems (D, N_1) and (D, N_2), when N_1 and N_2 are (L, ϵ)-close. Note that this distance is unbounded in general; hence, we focus on a bounded number of steps, i, of the executions of the system. The deviations in the states increase according to a geometric progression; we use the bounds on

Manuscript received 13 August 2024; accepted 13 August 2024. This work was supported in part by NSF under Grant 2008957 and an Amazon Research Award. This article was presented at the International Conference on Embedded Software (EMSOFT) 2024 and appeared as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: P. Habeeb.)*

P. Habeeb is with the Department of Computer Science And Automation, Indian Institute of Science (Bengaluru), Bengaluru 560012, India (e-mail: habeebp@iisc.ac.in).

Lipsy Gupta and Pavithra Prabhakar are with the Department of Computer Science, Kansas State University, Manhattan, KS 66506 USA (e-mail: lipsy@ksu.edu; pprabhakar@ksu.edu).

Digital Object Identifier 10.1109/TCAD.2024.3445813

geometric series to provide a bound on the distance between the states at the kth step of the closed-loop system.

The remaining part of the framework is to consider the problem of establishing (L, ϵ)-conformance between N_1 and N_2. Next, we consider the problem of computationally checking whether the NNs N_1 and N_2 are (L, ϵ)-conformant. We construct a new NN N, which outputs pairs (δ, γ), where δ captures the difference in inputs to the N_1 and N_2 and γ the corresponding difference in outputs. Hence, we reduce conformance checking problem to a reachability problem, wherein we check if $L\delta + \epsilon > \gamma$ for any pair (δ, γ) in the output set of the combined NN. This enables the use of existing NN verification tools [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] for conformance checking. Our experimental results on an autonomous rocket landing system demonstrate the feasibility of checking approximate conformance on different NNs trained for the same dynamics, as well as, the practical semantic closeness exhibited by the corresponding closed-loop systems.

The main contributions of this work are as follows.

1) We propose the notion of (L, ϵ)-conformance between two NNs, a notion of distance between two neural networks, that allows us to bound the distance in the reachable sets of the closed-loop systems in which they are deployed.
2) We provide a theoretical bound on the distance between states in the kth step of execution of two closed-loop systems with the same plant but different NN controllers that are (L, ϵ)-conformant.
3) We provide a method for checking (L, ϵ)-conformance between two neural networks, which includes a construction that merges the input networks into a single network and then reduces the (L, ϵ)-conformance to a reachability analysis.
4) We provide an experimental evaluation of the (L, ϵ)-conformance checking method on a set of neural networks trained for an automatic rocket landing case study. We report how conformance checking time is affected by the size of the network, investigate how the amount of perturbation in the networks changes the values of ϵ for which the network pairs are (L, ϵ)-conformant, and compare the theoretical bound and the actual state deviation between systems after k steps of execution.

II. Preliminaries

Let $\mathbb{R}$ denote the set of real numbers, and $\mathbb{N}$ denote the set of natural numbers. Given a non-negative integer k, let $[k]$ denote the set $\{0, 1, \ldots, k\}$, and (k) denote the set $\{1, \ldots, k\}$. ReLU activation function is defined as $\mathrm{ReLU}(x) = \max(0, x)\ \forall x \in \mathbb{R}$. For any set S, a valuation over S is a function $v : S \to \mathbb{R}$. We define $\mathrm{Val}(S)$ to be the set of all valuations over S. Let $n \in \mathbb{N}$. For $x \in \mathbb{R}^n$, let x_i denote the projection of x onto ith component, that is, $x = (x_1, x_2, \ldots, x_n)$. For $x \in \mathbb{R}^n$, the one norm on x is defined as $\|x\| = \sum_{i=1}^{i=n} |x_i|$. For a matrix $A \in \mathbb{R}^{n\times n}$, $a_{i,j}$ represents the elements in the ith row and jth column, and the one norm is defined as $\|A\| = \max_{1\leq j\leq n}(\sum_{i=1}^{i=n} |a_{i,j}|)$. Given two sets S_1 and S_2, the operation $\uplus$ represents the ordered union of S_1 and S_2. For $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$, $x \diamond y = (x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathbb{R}^{n+m}$. Given two functions $f : A \to B$ and $g : B \to C$, the composition of f and g, $g \circ f : A \to C$ is given by $g \circ f(a) = g(f(a))$.

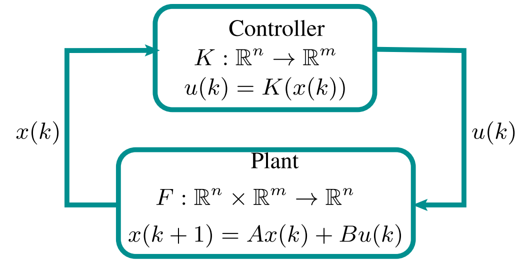


Fig. 1. Closed-loop system.

III. Closed-Loop Systems

In this section, we give a brief overview of closed-loop systems, and introduce a framework for conformance checking of closed-loop systems. A traditional closed-loop system consists of two components, namely, the plant that captures the dynamics of the physical system being controlled, and a controller that senses the state of the plant and computes actuator inputs, as shown in Fig. 1. We consider discrete-time systems, where the system evolves for one unit time in each iteration of the loop. Next, we formalize this.

Definition 1: A closed-loop system is a pair $S = (F, K)$, where 1) $F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ models the dynamics of the plant, $n \in \mathbb{N}$ and $m \in \mathbb{N}$ represent the dimensions of the state and the input vector, respectively, and 2) $K : \mathbb{R}^n \to \mathbb{R}^m$ represents the controller.

The controller senses the current state of the plant and outputs the control/actuation value to be input to the plant. The plant function F takes the current state of the system and the control input as its inputs, and outputs the state of the plant at the end of one time unit. The system starts from a given initial state $x(0) = x_0$. The feedback controller K takes the initial state of the system x_0 and outputs an input value $u_0 = K(x(0))$ to the plant. The plant then takes the input and evolves for a unit of time using the plant dynamics, reaching the next state $x_1 = F(x_0, u_0)$. This process repeats with the new state of the system, resulting in a sequence of states that we refer to as an execution.

Let $X_0 \subseteq \mathbb{R}^n$ be the sets of initial states, a sequence $\eta = x_0, x_1, x_2, \ldots,$ is called an execution of S from X_0, if there exists a sequence $u_0, u_1, u_2, \ldots,$ such that the following holds: 1) $x_0 \in X_0$, $u_0 = K(x_0)$ and 2) for each $i \geq 1$, $x_i = F(x_{i-1}, u_{i-1})$, and $u_i = K(x_i)$.

For an execution η, we use $\eta[i]$ to denote its ith element, that is, $\eta[i] = x_i$, and thus the reachable set of the above closed-loop is defined as follows. For each $i \geq 1$, the set

$$\mathrm{Reach}_S(X_0, i) = \{\eta[i] : \eta \text{ is an execution of } S \text{ from } X_0\}$$

is called the reachable set of the system at the ith step starting from the initial state X_0.

A. (L, ϵ)-Conformance of Closed-Loop Systems

Our broad objective is to investigate whether two closed-loop systems are behaviorally equivalent in the presence of evolving controllers, which is often the case when neural networks are deployed for control. However, it is often restrictive to assume that the controllers are behaviorally equivalent, and hence, we allow some deviations in the controllers. While equivalent controllers lead to exactly the same closed-loop behaviors, slight deviations in the controller behaviors can lead to nontrivial deviations in the closed-loop system behaviors. In this section, we aim to quantify this deviation and set the stage for applying it to NN controllers.

The classical notion of closeness for NN controllers [1], [2], [3], [4] studies the notion of ϵ-conformance, that requires the outputs of two networks to be within ϵ when given the same input. Note that this notion of conformance does not suffice for controllers in a closed-loop, since the ϵ deviations in the outputs of the controllers are fed back to the controllers in the next iteration and we need to bound the outputs of the controllers in the presence of small deviations in the input. Hence, we need a notion of conformance, that states that if the inputs deviate by at most δ, then the outputs deviate by at most $g(\delta)$, for some function g.

To obtain the form of g, we note that the controllers we consider are compositions of linear functions and activation functions. Let us consider the simple case where the two controllers are linear, that is, $K_1(x) = A_1x + B_1$ and $K_2(x) = A_2x + B_2$, where $x \in \mathbb{R}^n$, $A_1, A_2 \in \mathbb{R}^{m\times n}$ and $B_1, B_2 \in \mathbb{R}^{m\times 1}$. Note that

$$\begin{aligned}\|K_1(x_1) - K_2(x_2)\| &= \|A_1x_1 + B_1 - A_2x_2 - B_2\|\\ &= \|A_1x_1 + B_1 - (A_1 + D)x_2 - B_2\|\\ &\leq \|A_1(x_1 - x_2)\| + \|Dx_2\| + \|B_1 - B_2\|\\ &\leq \|A_1\|\|x_1 - x_2\| + \|Dx_2\| + \|B_1 - B_2\|.\end{aligned}$$

Assuming D is small (the deviation between A_1 and A_2), the output deviation can be some L times the input deviation plus an additive term ϵ. Inspired by this, we define the notion of (L, ϵ)-conformance, that stipulates that the outputs are within $L\delta + \epsilon$, when inputs deviate by at most δ.

Definition 2: Let $K_1, K_2 : A \to \mathbb{R}^m$, where $A \subseteq \mathbb{R}^n$, be two functions representing the controllers. For given $L > 0$ and $\epsilon > 0$, K_1 and K_2 are said to be (L, ϵ)-conformant if $\|K_1(x_1) - K_2(x_2)\| \leq L\|x_1 - x_2\| + \epsilon \;\forall x_1, x_2 \in A$.

B. Quantifying Closed-Loop Behavior for (L, ϵ)-Conformance

We quantify the behavior of two closed-loop systems with the same plant dynamics, but different controllers that are (L, ϵ)-conformant for some $L, \epsilon > 0$. In order to do this, we consider, in particular, a discrete-time linear dynamical system for the plant $S = (F, K)$ given by

$$x(k + 1) = Ax(k) + Bu(k) \qquad x(0) \in X_0 \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ are time invariant system matrices, and $u(0) = K(x_0)$. The following result gives a bound on the deviation of the kth elements of the executions of the two closed-loop systems which evolve through the above dynamics, have (L, ϵ)-conformance controllers, and when initiated from the same initial values.

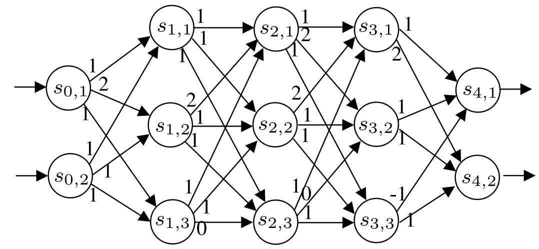


Fig. 2. Example NN N_1.

Theorem 1: Let $S_1 = (F, K_1)$ and $S_2 = (F, K_2)$ be two closed-loop systems with state and input dimensions, n and m, respectively, plant dynamics as described in (1), and the controllers K_1 and K_2 that are (L, ϵ)-conformant for some $L, \epsilon > 0$. For each $i \geq 1$, let $\eta_1[i]$ and $\eta_2[i]$ denote the ith elements of the executions of S_1 and S_2, respectively, starting from the state $\eta_1[0] = \eta_2[0] = x_0$. Then the following holds for all $k \geq 1$:

$$\|\eta_1[k] - \eta_2[k]\| \leq \|B\epsilon\| \frac{\left(1 - (\|A\| + \|B\|L)^k\right)}{1 - (\|A\| + \|B\|L)}. \tag{2}$$

If $\|A\| + \|B\|L = 1$, then $\|\eta_1[k] - \eta_2[k]\| \leq \|B\epsilon\|k$

Proof: (See Appendix for proof.) ■

IV. Neural Networks as Controllers

Our objective is to consider controllers that are neural networks. In this section, we provide preliminaries on neural networks, including definitions and semantics.

Definition 3: A NN is a tuple $N = (k, Act, \{S_i\}_{i\in[k]}, \{W_i\}_{i\in(k]}, \{B_i\}_{i\in(k]}, \{\sigma_i\}_{i\in(k]})$, where 1) $k \in \mathbb{N}$ represents the number of layers (except the input layer); 2) Act is a set of activation functions, and every $f \in Act$ has $\mathbb{R}$ as its domain and range; 3) $\forall\, i \in [k], S_i$ is a set of neurons of layer i, and $\forall\, i \neq j, S_i \cap S_j = \emptyset$; 4) $\forall\, i \in (k], W_i : S_{i-1} \times S_i \to \mathbb{R}$ is the weight function that captures the weights on the edges between the neurons at layer $i - 1$ and i; 5) $\forall\, i \in (k], B_i : S_i \to \mathbb{R}$ is the bias function that associates a bias value with neurons of layer i; and 6) $\forall\, i \in (k], \sigma_i : S_i \to Act$ is an activation association function that associates an activation function with each neuron of layer i.

The layers S_0 and S_k are called the input and the output layers, respectively; the other layers are said to be hidden. We fix the following notations for the rest of this article. Any NN denoted by N is the network $N = (k, Act, \{S_i\}_{i\in[k]}, \{W_i\}_{i\in(k]}, \{B_i\}_{i\in(k]}, \{\sigma_i\}_{i\in(k]})$ and for any $j \in \mathbb{N}$, $N_j = (k_j, Act, \{S_i^j\}_{i\in[k_j]}, \{W_i^j\}_{i\in(k_j]}, \{B_i^j\}_{i\in(k_j]}, \{\sigma_i^j\}_{i\in(k_j]})$. For notational convenience, we simplify the notation assuming the values that i iterates over are clear from the context. For example, for any $j \in \mathbb{N}$, we will write a NN N_j with k_j layers as $N_j = (k_j, Act, S_i^j, W_i^j, B_i^j, \sigma_i^j)$.

As an example, consider the NN N_1 in Fig. 2; it consists of an input layer with two neurons, three hidden layers with three neurons each, and an output layer with two neurons. The weights on the edges are shown, the biases are zero, and the activation functions are all ReLU.

Next, we define the executions of a NN as a sequence of valuations, each of which corresponds to values assigned to the neurons in a layer. Given a valuation v for a layer $i-1$, $[\![N]\!]_i(v)$ denotes the valuation obtained for the layer i according to the semantics of N, which is defined below.

Definition 4 (Semantics of a Neural Network): Given a NN N, the semantics of the layer i, $i \neq 0$, is the function $[\![N]\!]_i$: $\mathrm{Val}(S_{i-1}) \to \mathrm{Val}(S_i)$, where for any $v \in \mathrm{Val}(S_{i-1})$, $[\![N]\!]_i(v) = v'$, is given by

$$\forall s' \in S_i,\ v'(s') = \sigma_i(s')\left(\left(\sum_{s \in S_{i-1}} W_i(s, s')v(s)\right) + B_i(s')\right).$$

We define the semantics of NN N by the function $[\![N]\!] : \mathrm{Val}(S_0) \to \mathrm{Val}(S_k)$ as a composition of functions corresponding to individual layers as $[\![N]\!] = [\![N]\!]_k \circ [\![N]\!]_{k-1} \ldots \circ [\![N]\!]_1$.

For the input valuation $v(s_{0,1}) = 1$ and $v(s_{0,2}) = -1$, the NN in Fig. 2 gives the output valuation as $v(s_{4,1}) = 8$ and $v(s_{4,1}) = 24$.

Let us fix some more notations for the rest of this article. For a NN N, for each $i \in [k]$, let $S_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,r_i}\}$, and for a NN N_j, for each $i \in [k_j]$, let $S_i^j = \{s_{i,1}^j, s_{i,2}^j, \ldots, s_{i,r_{ij}}^j\}$. Since any valuation $v \in \mathrm{Val}(S_i)$ is a map $v : S_i \to \mathbb{R}$, and we have defined the ordering of the nodes in every layer, from now on we consider $v \in \mathrm{Val}(S_i)$ as an element of $\mathbb{R}^{|S_i|}$. Also, since our aim is to compare the behavior of two closed-loop systems with the same plant dynamics, but different neural networks as their controllers, all the neural networks are assumed to have same number of nodes in their input and output layers. In particular, for neural networks N, N_1, N_2, $|S_0| = |S_0^1| = |S_0^2| = n$ and $|S_k| = |S_{k_1}^1| = |S_{k_2}^2| = m$. Hence, the semantic of a NN can be considered as the following function $[\![N]\!] : \mathbb{R}^n \to \mathbb{R}^m$.

V. Verifying (L, ϵ)-Conformance of Neural Networks

In this section, we present our approach to check (L, ϵ)-conformance between two neural networks. First, we formally define the (L, ϵ)-conformance problem for neural networks.

Problem 1: Given two neural networks N_1 and N_2, two real numbers $\{L, \epsilon\} \subseteq \mathbb{R}_{>0}$, and a set of inputs $\mathcal{I} \subseteq \mathbb{R}^n$, the (L, ϵ)-conformance problem involves verifying whether, for all $v_1, v_2 \in \mathcal{I}$, the condition $\|[\![N_1]\!](v_1) - [\![N_2]\!](v_2)\| < L\|v_1 - v_2\| + \epsilon$ holds.

The broad idea to verify (L, ϵ)-conformance between two neural networks is to transform the problem into reachability analysis and utilize the existing reachability tools. Given two neural networks, N_1 and N_2, we construct a new NN $N_3 = \mathrm{merge}(N_1, N_2)$ which takes as inputs the inputs of the two neural networks v_1 and v_2, respectively, and outputs pairs (δ, γ), where δ is the norm of the difference in the inputs, that is, $\delta = \|v_1 - v_2\|$, and γ is the norm of the difference in the outputs, that is, $\gamma = \|[\![N_1]\!](v_1) - [\![N_2]\!](v_2)\|$. We need to check if $\gamma < L\delta + \epsilon$ for every (δ, γ) output by the new network. We check if the intersection of the constraint $\gamma \geq L\delta + \epsilon$ with the reachable set of this new network which captures all such (δ, γ) pairs, is unsatisfiable. The two networks are (L, ϵ)-conformant if the above condition is unsatisfiable, otherwise, they are not (L, ϵ)-conformant.

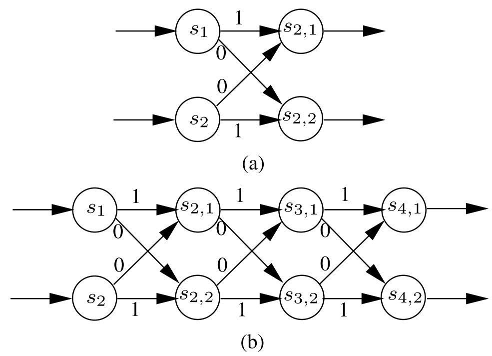


Fig. 3. Identity gadget IG example.

A. Identity and Difference Neural Networks

First, we define two kinds of gadgets that we use in the construction of the merged network $\mathrm{merge}(N_1, N_2)$.

1) Identity Gadget (IG): One gadget we need is a NN $\mathrm{IG}(n, r)$ with n input neurons that captures the identity relation and is of length r, that is, we need the NN to output the same values as its input. Fig. 3(a) shows a 1-layer NN with this property for two input neurons. We can repeat this structure r times to compute an r-layer NN with identity input-output relation. We assume that the inputs are all positive, and use ReLU functions as the activation function for all the nodes. Note that this NN can be appended to any NN layer with ReLU activation function without changing its input output semantics. As an example, consider a one-layer NN $\mathrm{IG}(2, 1)$ shown in Fig. 3(a) and a repetition length $r = 3$. The result of 3 repetitions given by $\mathrm{IG}(2, 3)$ is depicted in Fig. 3(b).

2) Difference Gadget (DG): The next network we require is one which takes as inputs $x_1 \diamond x_2$, where x_1 and x_2 have the same dimension n, and outputs $\|x_1 - x_2\|$. This is realized by a 2-layer gadget with $2n$ input nodes, $2n$ nodes in hidden layer and 1 output node. More generally, the network $\mathrm{DG}(n)$ has as input layer with two sets of nodes $S_1 = \{s_1^1, \ldots, s_n^1\}$ that takes x_1 as input and $S_2 = \{s_1^2, \ldots, s_n^2\}$ that takes x_2 as input. The hidden layer corresponds to nodes which compute the difference between the values of ith nodes from S_1 and S_2. For each i, there are two nodes, one of which computes $v(s_i^1) - v(s_i^2)$, and the other computes $v(s_i^2) - v(s_i^1)$. The output layer corresponds to summing up the values of the nodes in hidden layer. All of the activation functions are ReLU. So, only one of the values among $v(s_i^1) - v(s_i^2)$ and $v(s_i^2) - v(s_i^1)$, i.e., the positive one contributes to the summation, thereby computing the 1-norm at the output.

Fig. 4 shows such a network for dimension $n = 2$. Here, $S_1 = \{s_1, s_2\}$ and $S_2 = \{s_1', s_2'\}$. The nodes in the hidden layer correspond $v(s_1) - v(s_1')$, $v(s_1') - v(s_1)$, $v(s_2) - v(s_2')$, and $v(s_2') - v(s_2)$. The output corresponds to summing up the values $ReLU(v(s_1) - v(s_1')) + ReLU(v(s_1') - v(s_1)) + ReLU(v(s_2) - v(s_2')) + ReLU(v(s_2') - v(s_2))$.

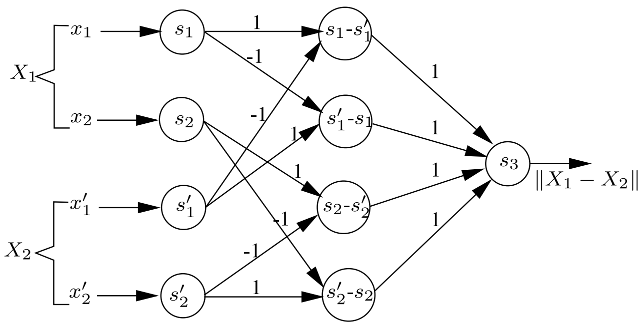
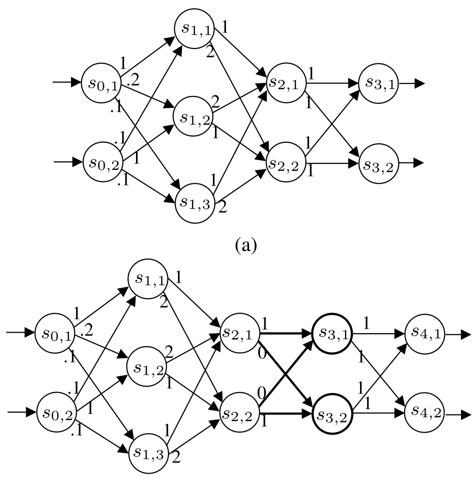


Fig. 4. Difference gadget (DG) example.



(a)

(b)

Fig. 5. Append example. (a) Neural network $N2$. (b) Appended network append(N_2, 1).

B. Merge Networks

Now, we explain the construction of the merging of the two neural networks to obtain N_3 = merge(N_1, N_2). The output network N_3 has two nodes in the output layer, denoted by s' and s, respectively, and the nodes in the input layer are the ordered union of the input nodes from the given input networks. Recall that for verifying the (L, ϵ)-conformance between N_1 and N_2, we aim to compute the values $\|v_1 - v_2\|$ and $\|[\![N_1]\!](v_1) - [\![N_2]\!](v_2)\|$ for all $v_1, v_2 \in \mathcal{I}$, where $\mathcal{I} \subseteq \mathbb{R}^n$. We construct the output network N_3 in such a way that the first output produces the value $\|v_1 - v_2\|$, and the second output produces the value $\|[\![N_1]\!](v_1) - [\![N_2]\!](v_2)\|$. In this construction, we utilize the previously defined gadgets.

First, we ensure that both networks have the same number of layers by using an operation called *append*, facilitated by our IG gadget. The formal definition of this function can be seen in Appendix. For a given NN N and a positive integer r, the *append* function, denoted as append(N, r), produces another NN by appending IG(l, r) to the last hidden layer of N where l is the number of neurons in this layer. We assume that all the activation functions for the hidden layers are ReLU, in which case, inserting the IG(l, r) gadget preserves the semantics. As an example of the *append* function, Fig. 5(b) illustrates the NN resulting from append(N_2, 1), where N_2 refers to the NN shown in Fig. 5(a). In our construction of the merged network merge(N_1, N_2), we first perform append(N, r) on the NN N with fewer number of layers among N_1 and N_2, where r is the difference in the number of layers between the two networks.

Now, let's explain the *merge* function at a high level using an example illustrated in Fig. 6, where we combine networks N_1 (depicted in Fig. 2) and N_2 [depicted in Fig. 5(b)] into a single network, denoted as N_3. The formal definition of this function can be found in Appendix.

Let N_1 and N_2 each have n inputs and m outputs. We apply our difference gadget DG(m) to the output layers of N_1 and N_2 to compute the one norm of the difference of the outputs of N_1 and N_2; let us call the node capturing the difference s. Similarly, we apply our difference gadget DG(n) to the input layers of N_1 and N_2 to compute the one norm of the difference of the inputs to N_1 and N_2; let us call the node capturing the difference to be t. Note that we want the values at t to be an output along with s. Hence, we append IG$(1, r)$ to t for an appropriate r (the difference in the layer number of s and t).

As illustrated in Fig. 6, the first node, s'_2, in the third layer of N_3 outputs the one norm of the input differences. We utilize the identity gadget IG to propagate the one norm of the input differences to the output layer of the merged network. The output node s' produces the one norm of the input differences. Similarly, to compute the one norm of the output differences, DG is appended to the output nodes of the two networks. As shown in Fig. 6, the node s in the output layer computes the one norm of the output differences.

From the construction of the merged network, we can easily see that the output node s' computes $\|v_1 - v_2\|$, while the output node s computes the value $\|[\![N_1]\!](v_1) - [\![N_2]\!](v_2)\|$, where v_1 and v_2 are inputs given to N_1 and N_2, respectively. Formally we have the following result.

Proposition 1: Given two neural networks, N_1 and N_2, let merge(N_1, N_2) = N_3. Then $\forall\, v_1, v_2 \in \mathbb{R}^n$, $[\![N_3]\!](v_1 \diamond v_2) = (\|v_1 - v_2\|, \|[\![N_1]\!](v_1) - [\![N_2]\!](v_2)\|)$.

C. Reachability-Based Approach for (L, ϵ)-Conformance Verification

In this section, we explain our approach to check the (L, ϵ)-conformance between two neural networks. This approach is based on the reachability analysis of the merged network that we constructed in the previous section. The reachable set of a NN for a given set of input values is the set of output values that we obtain through the network. We define the reachable set formally as follows.

Definition 5: The reachable set of a NN N w.r.t a set $\mathcal{I} \subseteq \mathbb{R}^n$ is

$$\mathcal{R}_N(\mathcal{I}) = \{[\![N]\!](v) | v \in \mathcal{I}\}.$$

We use the merged network generated using the merge procedure defined in the previous section to check the conformance between networks. From the merge construction, we can see that if N_3 = merge(N_1, N_2), then for the input vector (v_1, v_2), the output node s' of N_3 outputs $\|v_1 - v_2\|$ while the output node s computes the value $\|[\![N_1]\!](v_1) - [\![N_2]\!](v_2)\|$. So, the networks N_1 and N_2 satisfy the (L, ϵ)-conformance property for a given input valuation if and only if the value

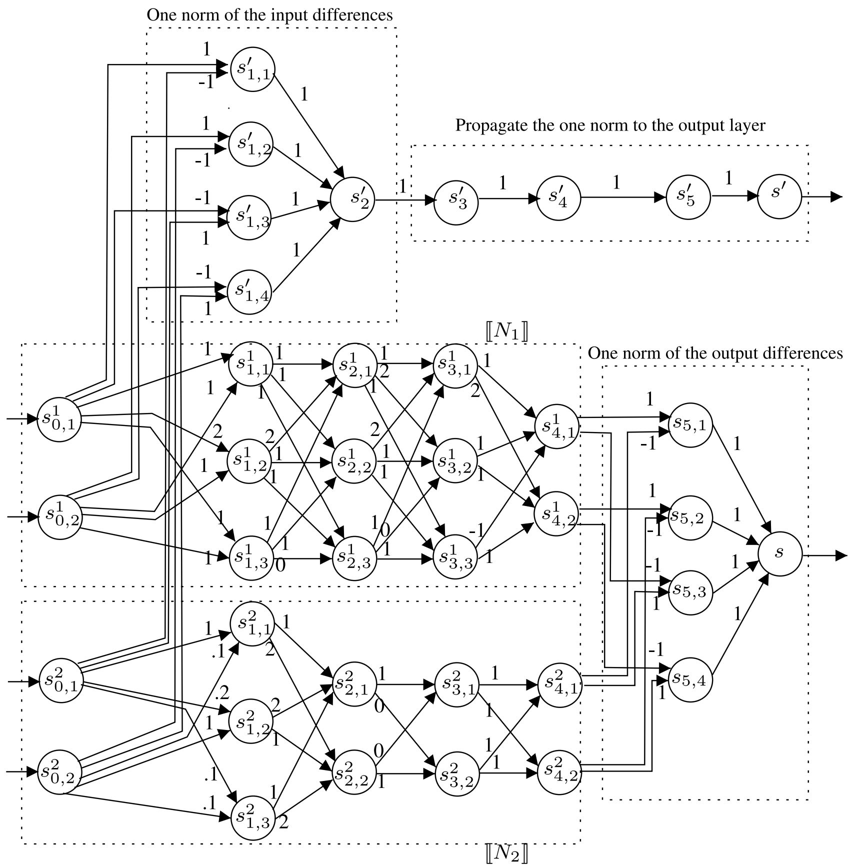


Fig. 6. Merged network $N_3 = \text{merge}(N_1, N_2)$.

of the output node s is less than the result of multiplying the value of s' by L and adding ϵ to it.

Theorem 2: Given two neural networks N_1 and N_2, two real numbers $L > 0$ and $\epsilon > 0$, and a set of inputs $\mathcal{I} \subseteq \mathbb{R}^n$, N_1 and N_2 satisfy (L, ϵ)-conformance property if and only if $\gamma < L\delta + \epsilon \quad \forall(\delta, \gamma) \in \mathcal{R}_{N_3}(\mathcal{I}')$, where $N_3 = \text{merge}(N_1, N_2)$, $\mathcal{I}' = \{v_1 \diamond v_2 \mid v_1, v_2 \in \mathcal{I}\}$.

We use the *nnenum* tool [5] to check the (L, ϵ)-conformance. The tool nnenum [5] is a star-set-based reachability analysis tool which incorporates abstraction refinement and optimization techniques to accelerate the reach set computation. The tool takes, as input, the merged network, interval values for each input node, and the property to be checked. The property to be checked is whether the value of the second output node, which is the one norm of the output differences of the networks for the given input values, is greater than or equal to the first output value multiplied by L, and then ϵ is added to it. The tool outputs either 'sat' or 'unsat'. If the output is 'sat', then there exists an input value that violates the (L, ϵ)-conformance property. If it is 'unsat', then the input networks satisfy the property for the values in the given input intervals.

We can use any of the existing NN tools to check (L, ϵ)-conformance between two neural networks. Several NN verification methods have been explored in the literature, including those based on SMT solving, such as Reluplex [8] and Marabou [6], those based on MILP, such as Sherlock [16] and alpha-beta-CROWN [11], and those based on abstract interpretation, such as AI^2 [15] and DeepPoly [17].

VI. Experiments

In this section, we describe our experimental setup, including the autonomous rocket landing case study, the results of evaluating our (L, ϵ)-conformance approach on a set of neural networks trained for the case study, and the results of comparison of the actual and the theoretical deviation of the system states after k steps for different pairs of the networks. All experiments were conducted on an Ubuntu machine with an Intel Core i5-10210U 1.60 GHz CPU and 8GB RAM.

A. Case Study—Autonomous Rocket Landing System

We start by explaining our autonomous rocket landing case study. In this system, the rocket's internal control system hands over the maneuver task to the automatic landing system at a particular height from the ground. The automated landing system aims to reduce the rocket's velocity by applying thrust and thereby, achieves a smooth landing on the ground.

1) System Dynamics: The system's state is a 2-D real-valued vector $[p, v]^T$, representing the rocket's position (distance from the ground) and velocity. The NN controller takes the system's current state and outputs the amount of thrust that should be applied to the system. The following equation describes the dynamics of the plant:

TABLE I
(L, ϵ)-CONFORMANCE VERIFICATION

$L, \epsilon \rightarrow$		1,0.5		10,0.5		100,0.5		1,2		10,2		100,20	
Net1	Net2	C?	Time	C?	Time	C?	Time	C?	Time	C?	Time	C?	Time
N_1	N_2	No	5s	No	5s	No	5s	Yes	185m	Yes	184m	Yes	172m
N_1	N_3	No	7s	No	7s	No	6s	Yes	351m	Yes	355m	Yes	349m
N_1	N_4	No	8s	No	8s	No	7s	Yes	477m	Yes	441m	Yes	530m
N_1	N_5	No	9s	No	10s	No	9s	Yes	580m	Yes	561m	Yes	539m
N_2	N_3	No	7s	No	7s	No	6s	Yes	315m	Yes	309m	Yes	304m
N_2	N_4	No	9s	No	9s	No	8s	Yes	535m	Yes	442m	Yes	443m
N_2	N_5	No	11s	No	11s	No	11s	Yes	528m	Yes	544m	Yes	544m
N_3	N_4	No	7s	No	7s	No	7s	Yes	493m	Yes	490m	Yes	497m
N_3	N_5	No	9s	No	8s	No	8s	Yes	599m	Yes	605m	Yes	618m
N_4	N_5	No	8s	No	8s	No	7s	Yes	519m	Yes	515m	Yes	525m

$$x(k+1) = Ax(k) + Bu(k) \tag{3}$$

where $x(k) = [p, v]^T$ represents the position and velocity at time k and $u(k)$ is the output of the NN at step k. We are considering two systems with different system matrices. The first system has the following matrices:

$$A = \begin{bmatrix} 1 & -\tau \\ 0 & 1 \end{bmatrix}, \text{ and } B = \begin{bmatrix} 0 \\ -100\tau \end{bmatrix} \tag{4}$$

and the second system has the following matrices:

$$A = \begin{bmatrix} 0.8 & -\tau \\ 0 & 0.8 \end{bmatrix}, \text{ and } B = \begin{bmatrix} 0 \\ -100\tau \end{bmatrix} \tag{5}$$

where $\tau = 0.001$ is the sampling period. These two systems can have considerably different theoretical upper bounds on the state deviations given by Theorem 1. In the first system, we have $\|A\| = 1.001$ and $\|B\| = 0.1$. Since $\|A\| > 1$, no matter what value of L we take, $\|A\| + \|B\|L$ will always be greater than 1, which results in large theoretical upper bounds on the state deviation. Since we would also like to study the case where we have the possibility of $\|A\| + \|B\|L < 1$ for some L, we consider another system with the system matrices shown in (5), where $\|A\| = 0.801$ and $\|B\| = 0.1$.

2) Neural Network Controller: We use the deep deterministic policy gradient (DDPG) method [18] to train reinforcement-learning-based NN controllers. To generate training scenarios, we implement the rocket landing system as an environment in the OpenAI Gym toolkit [19]. The initial position and velocity of the rocket are randomly chosen from the intervals $[50, 60]$ and $[40, 55]$, respectively. The termination condition corresponds to either velocity being ≤ 0 or position being ≤ 0. The reward function $\rho(p, v)$ considered is given below

$$\begin{aligned} \rho(p, v) = -(p+v) \ &+ I_1(p, v) + I_2(p, v) \\ &+ r(\text{safeLanding}) + r(\text{collision}). \end{aligned}$$

Here, $I_1(p, v)$ returns a reward of 20 if $p \in [3, 6] \wedge v \in [2, 5]$; returns a reward of 0 if $p \notin [3, 6] \wedge v \notin [2, 5]$, otherwise it returns a reward of -100. Similarly, $I_2(p, v)$ returns a reward of 20 if $p \in [1, 3) \wedge v \in [1, 2)$, returns a reward of 0 if $p \notin [1, 3) \wedge v \notin [1, 2)$, otherwise it returns a reward of -100. $r(\text{safeLanding})$ returns a reward of 10 000 if $p = 0 \wedge v = 0$, and r(collision) returns a reward of -500 if $(p{<}0 \wedge v{>}0) \vee (p > 0 \wedge v <= 0)$.

We train a total of five networks for 100 epochs each; details of the architecture of each of the networks are as follows: N_1 (2, 500, 400, 300, 1), N_2 (2, 500, 400, 1), N_3 (2, 500, 400, 300, 200, 1), N_4 (2, 500, 400, 300, 200, 100, 1), and N_5 (2, 500, 400, 300, 200, 100, 100, 1), where the numbers in parenthesis represent the number of nodes in each layer.

B. Experiments 1—(L, ϵ)-Conformance Checking

In our first set of experiments, we evaluate our (L, ϵ)-conformance checking approach on each possible pair of networks that we trained for our case study. We use the *nnenum* tool [5], [20] to check the conformance condition on the output of the merged NN. The tool takes the merged network, an interval of values for each input node, and the property for output nodes to check, and outputs whether the network satisfies the property for the given input interval.

We assess our approach for different values of L and ϵ. For the experiment, the input interval for the position is taken to be $[0, 60]$, and the input interval for velocity is taken to be $[0, 40]$ for both the networks. The property we check on the output reach set is whether the second part of the output corresponding to the norm of the difference between the NN outputs is greater than or equal to L times the first part of the output corresponding to the norm of the difference between the NN inputs plus ϵ. If the property is unsatisfiable, then the networks are (L, ϵ)-conformant as shown in Theorem 2.

Experimental results are shown in Table I. The first two columns represent the names of the networks. Each subsequent column is divided into two subcolumns. For each main column, we report the L and ϵ values. For example, the third column corresponds to $L = 1$ and $\epsilon = 0.5$. The subcolumn titled "C?" indicates whether the network pairs satisfy the conformance property ("Yes" means the network pairs satisfy the property). The "Time" subcolumn provides the time taken to check the property.

Now, we consider neural networks from the VCAS benchmark from ARCH-COMP AINNCS [21]. This benchmark is a closed-loop variant of the aircraft collision avoidance system ACAS X. The scenario involves two aircraft, the ownship and the intruder, where the ownship is equipped with a collision avoidance system. The network contains an input layer with 3 nodes, five hidden layers with 20 nodes each, and an output layer with 9 nodes (see [21] for more details on this benchmark). In this experiment, we evaluate our (L, ϵ)-conformance checking technique on eight pairs of networks from this benchmark for some (L, ϵ) values. The results are shown in Table II, where the first two columns are

TABLE II
(L, ϵ)-CONFORMANCE VERIFICATION ARCH-COMP VCAS BENCHMARK

$L, \epsilon \rightarrow$		1,0.5		1,10		1,100	
Net1	Net2	C?	Time	C?	Time	C?	Time
P_01	P_02	No	1s	Yes	1s	Yes	1s
P_01	P_03	No	1s	Yes	1s	Yes	1s
P_01	P_04	No	1s	Yes	1s	Yes	1s
P_01	P_05	No	1s	Yes	1s	Yes	1s
P_01	P_06	No	1s	Yes	1s	Yes	1s
P_01	P_07	No	1s	No	1s	Yes	1s
P_01	P_08	No	1s	Yes	1s	Yes	1s
P_01	P_09	No	1s	Yes	1s	Yes	1s

network pairs, and the remaining columns show whether the network pairs are conformant (Yes) or nonconformant (No). From this experiment, we can see that our approach is able to check conformance within a second for all the network pairs considered.

From Table I, we observe that in case that the network pairs are not conformant, the tool returns quickly, usually, within seconds. On the other hand, establishing conformance takes longer. Intuitively, to establish conformance, the tool needs to show that there are no satisfying assignments, while to show that they are not conformant, it just needs to find one (δ, γ) pair that violates the constraint. On the other hand, in case of conformant networks, we note that the conformance checking time is not affected much when we vary L and ϵ.

We now analyze how the values of L and ϵ depend on each other and how the theoretical bounds deviate for different pairs of L and ϵ. For this experiment, we consider a pair of networks (P_{01}, P_{02}) from the VCAS benchmark [21], and we determine different values of (L, ϵ) pairs for which the network pairs are conformant. Then we report the deviation of the systems with dynamics shown in (4) and (5) after 100 steps of execution. Results are shown in Table III, where the first two columns are the L and ϵ values for which the network pair is checked for conformance, and the "Time" column shows the time taken to check the conformance. The last two columns show the theoretical upper bound on the state deviation after 100 execution steps using the expression given in Theorem 1 for systems with dynamics given by (4) and (5), respectively. Notice that the value of theoretical bound tends to increase with L, when $\|A\| + \|B\|L > 1$, while the effect is minimal when $\|A\| + \|B\|L < 1$. Hence, we choose an arbitrary L, say $L = 1$, in the rest of the experiment.

We now investigate how the conformance checking time is affected by the size of the merged network. For this experiment, we train networks with two input nodes, one output node, and five hidden layers, each with varying numbers of nodes, using the same training settings explained above. Network N_5 has five nodes in each hidden layer, N_10 has 10 nodes in each hidden layer, similarly, networks N_50, N_100, and N_500 have 50, 100, and 500 nodes in each hidden layer, respectively. We fix $L = 1$ and check conformance for different values of ϵ, where input intervals for the two input nodes are [0, 60] and [0, 40], respectively. The results are shown in Table IV, where the first two columns represent the names of

TABLE III
L VERSUS ϵ, NETWORKS: P_01 AND P_02INPUT INTERVALS: $[-133, -129]$, $[-22.5, -19.5]$, AND $[24.5, 25{,}5]$

L	ϵ	C?	Time	∇_{100}	
				Eqn (4)	Eqn (5)
0.0001	2.84	No	2s	-	-
0.0001	2.85	Yes	2s	29.973	1.432
0.001	2.83	No	2s	-	-
0.001	2.84	Yes	2s	30.003	1.427
0.01	2.80	No	2s	-	-
0.01	2.81	Yes	2s	31.072	1.419
0.1	2.79	No	2s	-	-
0.1	2.80	Yes	2s	50.557	1.481
1	2.79	No	2s	-	-
1	2.80	Yes	2s	50.557	1.481
2	2.79	No	2s	-	-
2	2.80	Yes	2s	50.557	1.481

TABLE IV
NETWORK SIZE VERSUS CONFORMANCE CHECKING TIME. INPUT INTERVALS: [0,60] AND [0,40], $L = 1$

Net1	Net2	#Nodes in Merged Net	ϵ	C?	Time
N_5	N_10	4,19,16,16,16,16,3,3,2	2.5	No	1s
			3	Yes	7s
N_5	N_50	4,59,56,56,56,56,3,3,2	0.5	No	1s
			1	Yes	53s
N_10	N_50	4,64,61,61,61,61,3,3,2	1	No	1s
			3	Yes	3m 40s
N_5	N_100	4,109,106,106,106,106,3,3,2	0.5	No	2s
			0.8	Yes	2m 5s
N_10	N_100	4,114,111,111,111,111,3,3,2	2	No	2s
			3	Yes	5m 45s
N_5	N_500	4,509,506,506,506,506,3,3,2	0.5	No	6s
			1	Yes	14m 8s
N_50	N_500	4,554,551,551,551,551,3,3,2	0.1	No	6s
			0.5	Yes	159m 18s
N_100	N_500	4,604,601,601,601,601,3,3,2	0.1	No	7s
			0.5	Yes	223m 39s

the networks, and the third column gives the number of nodes in each layer of the merged network. The ϵ column represents the values of ϵ that are checked for conformance, C? indicates whether the network pairs are conformant for $L = 1$ and the corresponding ϵ. The final column titled "Time" provides the time taken to check conformance. From the results, we can see that the time required to prove that the networks are (L, ϵ)-conformant increases with the number of nodes in the merged network. As explained in the earlier experimental results, demonstrating that networks are nonconformant takes very little time, and it is not significantly affected by the size of the merged network.

In the next set of experiments, we investigate how the amount of perturbation in the networks changes the values of ϵ for which the network pairs are (L, ϵ)-conformant. For these experiments, we manually create a NN with four layers, each layer containing two nodes. Then, we create four more networks by perturbing the original network with different amounts of perturbation. The perturbation involves randomly adding or subtracting a given value to each of the weights and bias values of the original network. The results of these experiments are shown in Table V, where the first column displays the amount of perturbation, the second and third columns represent the values of L and ϵ, respectively, for which the conformance is checked, and the last column indicates whether the network pairs (the original network and the perturbed network) are conformant to each other or not.

TABLE V
PERTURBATION VERSUS ϵ INPUT INTERVALS: [0,10] AND [0,10]

Perturbation	L	ϵ	Conformance?
0.0001	1	1000	No
	1	1100	Yes
0.1	1	1000	No
	1	1100	Yes
1	1	1400	No
	1	1500	Yes
5	1	6000	No
	1	8000	Yes

TABLE VI
STATE DIFFERENCE, PLANT DYN = (4), AND INIT STATE $X_0 = (60, 40), L = 1, \epsilon = 2$

Net1	Net2	10	100	200	400	600
N_1	N_2	0.4807	4.3164	7.9958	13.8515	20.3119
N_1	N_3	1.1353	9.6470	17.5744	30.2440	41.9112
N_1	N_4	0.6410	5.7648	10.8378	19.3043	28.4606
N_1	N_5	1.7235	15.4416	28.6498	50.2338	69.4564
N_2	N_3	0.6545	5.3306	9.5786	16.3925	21.5993
N_2	N_4	0.1603	1.4484	2.8419	5.4528	8.1487
N_2	N_5	1.2427	11.1252	20.6540	36.3823	49.1445
N_3	N_4	0.4942	3.8821	6.7366	10.9397	13.4506
N_3	N_5	0.5882	5.7946	11.0754	19.9897	27.5451
N_4	N_5	1.0824	9.6767	17.8120	30.9295	40.9957
∇_k		3.20e+0	2.99e+04	4.51e+08	1.03e+17	2.34e+25

TABLE VII
STATE DIFFERENCE, PLANT DYN = (5), AND INIT STATE $X_0 = (60, 40), L = 1, \epsilon = 2$

Net1	Net2	10	100	200	400	600
N_1	N_2	5.16e-02	8.64e-10	3.20e-19	2.30e-38	1.26e-57
N_1	N_3	1.22e-01	1.92e-09	6.99e-19	4.93e-38	2.52e-57
N_1	N_4	6.88e-02	1.16e-09	4.36e-19	3.23e-38	21.65e-57
N_1	N_5	1.85e-01	3.10e-09	1.15e-18	8.24e-38	4.26e-57
N_2	N_3	7.00e-02	1.06e-09	3.79e-19	2.63e-38	1.26e-57
N_2	N_4	1.72e-02	2.94e-10	1.17e-19	9.30e-39	3.82e-58
N_2	N_5	1.33e-01	2.23e-09	8.27e-19	5.94e-38	3.00e-57
N_3	N_4	5.28e-02	7.62e-10	2.63e-19	1.70e-38	8.76e-58
N_3	N_5	6.35e-02	1.18e-09	4.47e-19	3.31e-38	1.74e-57
N_4	N_5	1.16e-01	1.94e-09	7.10e-19	5.01e-38	2.62e-57
∇_k		1.3079	2.0201	2.0202	2.0202	2.0202

TABLE VIII
L, ϵ AND STATE DEVIATION INPUT INTERVALS: $[-10, 10]$ AND $[-10, 10]$, $\|A\| = 0.5$, $\|B\| = 0.4$, AND INIT STATE $X_0 = (5, 5)$

L	ϵ	10	100	200	400	600
0.001	1190	9.52e+02	9.53e+02	9.53e+02	9.53e+02	9.53e+02
0.01	1180	9.51e+02	9.52e+02	9.52e+02	9.52e+02	9.52e+02
0.1	1180	1.02e+03	1.03e+03	1.03e+03	1.03e+03	1.03e+03
1	1200	3.13e+03	4.80e+03	4.80e+03	4.80e+03	4.80e+03
2	1200	2.05e+04	3.97e+14	9.84e+25	6.05e+48	3.72e+71
10	1200	4.67e+08	2.87e+67	6.02e+132	2.64e+263	inf
100	500	6.01e+16	2.82e+161	inf	inf	inf
1000	400	4.81e+16	2.25e+161	inf	inf	inf
$\triangle_k$		0.488	0.170	1.749	0.170	0.498

For this experiment, we fix the value $L = 1$. From the results, we can observe that the value of ϵ for which the networks satisfy the (L, ϵ)-conformance increases drastically with the amount of perturbation applied to the network.

C. Experiments 2—State Deviation Quantification

In our second set of experiments, we compare the actual difference in the state of the two systems after k steps of execution in practice with the difference computed using our theoretical bound presented in Theorem 1. The systems have the same plant, but different NN controllers. Both systems start in a given initial state; in our experiments, we use 60 as the initial position and 40 as the initial velocity of our case study system. To compute the actual difference in state after k steps, we run the systems separately for k steps and then note the deviation in the state. We compute the theoretical deviation in state by using the expression given in Theorem 1.

We use the same networks as before and the two variants of the plant dynamics given by (4) and (5). Results of the experiment are shown in Table VI for the system with dynamics given by (4), and Table VII for the system with dynamics given by (5). The first two columns in the tables show the names of the neural networks. Subsequent columns' titles show the number of steps after which we computed the actual state difference. The last row in the tables, starting with ∇_k, displays the approximate theoretical deviations of the systems after k steps. For example, the result obtained by substituting values into (2) for our first case study is 3.20×10^0, as shown in the last row of Table VI. Similarly, for the second system, the theoretical deviation after 10 steps is 1.3079, as shown in the last row of Table VII.

From these experiments, we observe that the actual deviation is consistently less than the theoretical bound computed using our formula in Theorem 1. However, the difference between actual deviation and theoretical deviation is less when $(\|A\| + \|B\|L)$ is less than 1 as compared to when it is greater than 1.

In the next set of experiments, we aim to evaluate how to choose the values of L and ϵ if the networks are (L, ϵ)-conformant to different values of L and ϵ. For this experiment, we fix two networks and determine different values of L and ϵ for which the networks are conformant. Subsequently, we analyze the practical and theoretical deviations in the state after a certain number of execution steps. We use the original network from Table V, and the network generated from this network by perturbing weights and biases randomly with values of either $+0.0001$ or -0.0001. The input intervals for these experiments are $[-10, 10]$ for both input nodes. We consider a system with the following system matrices:

$$A = \begin{bmatrix} 0.3 & -0.1 \\ 0.2 & 0.3 \end{bmatrix}, \text{ and } B = \begin{bmatrix} 0 \\ -0.4 \end{bmatrix} \tag{6}$$

where $\|A\| = 0.5$ and $\|B\| = 0.4$. We choose different values for L and determine values of ϵ such that the networks are (L, ϵ)-conformant to each other. The experimental results are shown in Table VIII, where the first two columns represent L and ϵ values for which the networks are conformant. The remaining columns display the theoretical bounds, given by Theorem 1, on the state deviation after a specified number of steps, as indicated in the column titles. The last row $\triangle_k$ shows the actual state deviation between systems after k execution steps starting from a state (5, 5). This experiment indicates that the tightest theoretical bound can be achieved in case the value of L is chosen as small as possible.

We summarize the observations from the experiments as follows. In the first set of experiments, we checked (L, ϵ)-conformance of a set of networks using our approach and found that the conformance checking procedure takes less time when the networks are not conformant, while it takes more time when they are conformant, with the execution time being largely unaffected by L and ϵ. We also observed that conformance checking time increases with the size of the merged network. Additionally, the value of ϵ increases significantly with the amount of perturbation applied to the network. In the second set of experiments, we compared the state difference in closed-loop systems after k execution steps with the theoretical bound from Theorem 1 and found that the actual state deviation is considerably less than the theoretical value. Finally, we determined that the tightest theoretical bound is achieved by choosing the smallest possible values for L and ϵ.

VII. Related Works

Safety verification of neural networks has gained prominence in recent years resulting in several tools, including those based on symbolic state-space exploration [5], [15], [17], [22] and constraint solving-based approaches [6], [8], [11], [16]. Several works also explore the safety of closed-loop systems with NN controllers [14], [23], [24], [25]. Despite the progress, scalability of the verification techniques remains a challenge.

Our focus here is on bounded safety verification of closed-loop systems with evolving neural networks. Our broad idea is to leverage the small changes in subsequent NN controllers by quantifying the change in terms of (L, ϵ)-conformance, using that to bound the distance between reach sets of corresponding closed-loop systems and transferring safety proofs to the newer versions. Notion of distance between systems has been extensively studied in the literature for different classes of systems. Classical notion of bisimulation [26] captures equivalence between processes. In the context of control and hybrid systems, this has been extended with approximate notions [27], [28] which ensure that the trajectories of two approximately bisimilar systems are within a bounded distance ϵ. The application of approximate bisimulation for reachability verification [29], [30], as well as algorithms for checking approximate bisimulation [27] have been explored.

The notion of ϵ-conformance has been studied for neural networks, wherein the output is assumed to be within a bound ϵ given the same input [1], [2], [3], [4], [31]. However, in the context of closed-loop systems, the inputs to the neural networks are not identical in different iterations through the loop, and hence, one needs a more general notion that provides bounds on output when given slightly different inputs. This is captured by notion of (L, ϵ)-conformance proposed. We note that even with this notion of (L, ϵ)-conformance between neural networks N_1 and N_2, the trajectories of the corresponding closed-loop systems are not guaranteed to be within ϵ. Instead, the deviation accumulates over iterations. This is in contrast to the notion of approximate bisimulation where the trajectories are bounded by ϵ. The reason for the ϵ-boundedness of the trajectories in approximate bisimulation arises due to the fact that it requires two states that are bisimilar to be within distance ϵ and the property to also hold after one transition. However, such a requirement is too strong to be satisfied by two neural networks, which translates to requiring the outputs of neural networks to be within ϵ, when given inputs that are within ϵ.

VIII. Conclusion

In this article, we explored the problem of approximate conformance checking of closed-loop systems with different neural networks controllers. We introduced the notion of (L, ϵ)-conformance between two neural networks, and used that to provide theoretical bounds on the closed-loop system behaviors. Further, we provided a technique for checking (L, ϵ)-conformance by reducing it to reachability analysis on a transformed network. Our experimental analysis demonstrates the feasibility of (L, ϵ)-conformance checking algorithm for two neural networks and the closeness of the resulting closed-loop system behaviors.

In this article, we assume the plant to be a discrete-time linear system. In the future, we would like to explore more complex and continuous time dynamics. Also, our experimental results show a large gap between the theoretical and practical deviations of the closed-loop system states, especially for the case when $(\|A\| + \|B\|L)$ is greater than 1. We would like to explore tighter theoretical bounds as it will be a crucial component of proof transfer using our (L, ϵ)-conformance method. For instance, if we have established that the reachable set of a system with a NN N up to k steps is at least α away from the unsafe set, and that the newer version of the NN N' is (L, ϵ)-conformant with N, we only need to check that the theoretical bound on the deviation between the closed-loop systems for this (L, ϵ)-conformance and k is less than α to deduce safety of the closed-loop system with N' as the controller. Hence. a tighter theoretical bound would be more successful at establishing safety.

Appendix

A. Proof of Theorem 1

Let $u_1^0, u_1^1, \ldots,$ and $u_2^0, u_2^1, \ldots,$ be the sequences of control inputs corresponding to the executions η_1 and η_2, respectively, that is, for each $i \geq 0$, $u_1^i = K_1(\eta_1[i])$ and $u_2^i = K_2(\eta_2[i])$. Also, for each $i \geq 0$, let $\triangledown_i = \|\eta_1[i] - \eta_2[i]\|$ and $\mu_i = \|u_1^i - u_2^i\|$. Since K_1 and K_2 are (L, ϵ)-conformant, $\mu_i \leq L\triangledown_i + \epsilon$. Note that $\triangledown_0 = 0$. Let $h = \|A\| + \|B\|L$, and $c = \|B\|\epsilon$. Then $\forall\, k \in \mathbb{N}$

$$
\begin{aligned}
\triangledown_k &= \|\eta_1[k] - \eta_2[k]\| \\
&= \|A\eta[k-1] + Bu_1^{k-1} - A\eta_2[k-1] - Bu_2^{k-1}\| \\
&\quad \text{(from system dynamics)} \\
&= \|A(\eta_1[k-1] - \eta_2[k-1]) + B\Big(u_1^{k-1} - u_2^{k-1}\Big)\| \\
&\leq \|A\|\triangledown_{k-1} + \|B\|\mu_{k-1} \\
&\leq \|A\|\triangledown_{k-1} + \|B\|(L\triangledown_{k-1} + \epsilon) \\
&\quad \text{since } \mu_{k-1} \leq L\triangledown_{k-1} + \epsilon \\
&= (\|A\| + \|B\|L)\triangledown_{k-1} + \|B\|\epsilon \\
&= h\triangledown_{k-1} + ch = \|A\| + \|B\|L, c = \|B\|\epsilon \\
&\leq h(h\triangledown_{k-2} + c) \ + c \\
&\leq h^k\triangledown_{k-k} + h^{k-1}c + h^{k-2}c + \cdots + c
\end{aligned}
$$

$$= c\left(h^{k-1} + h^{k-2} + \cdots + 1\right)\text{since } \nabla_0 = 0$$

$$= c\frac{\left(1 - h^k\right)}{1 - h}\text{for } h \neq 1$$

$$= \|B\|\epsilon\frac{\left(1 - (\|A\| + \|B\|L)^k\right)}{1 - (\|A\| + \|B\|L)}.$$

B. Formal Definition of the Append Function

Here, we provide the formal definition of the *append* function, which we utilized in the construction of the merged network.

Definition 6 (Append Hidden Layer): Given a NN $N_1 = (k_1, Act, S^1_i, W^1_i, B^1_i, \sigma^1_i)$ and an append count $r \in \mathbb{N}$, the append hidden layer function append(N_1, r) returns a NN $N_2 = (k_2, Act, S^2_i, W^2_i, B^2_i, \sigma^2_i)$, where

1) $k_2 = k_1 + r$;
2) for each $i \in [k_1 - 1]$, $S^2_i = S^1_i$,
   for each $i \in [k_1, k_2 - 1]$, $S^2_i = S^1_{k_1 - 1}$
   $S^2_{k_2} = S^1_{k_1}$;
3) for each $i \in (k_1 - 1]$, the functions W^2_i and W^1_i are the same,
   for each $i \in [k_1, k_2 - 1]$, $W^2_i(s^2_{i-1,l}, s^2_{i,m}) = 1$ if $l = m$, and 0 otherwise.
   $W^2_{k_2}(s^2_{k_2-1,i}, s^2_{k_2,j}) = W^1_{k_1}(s^1_{k_1-1,i}, s^1_{k_1,j})$;
4) for each $i \in (k_1 - 1]$, the functions B^2_i and B^1_i are the same,
   for each $i \in [k_1, k_2 - 1]$, B^2_i is a zero function,
   $B^2_{k_2}(s^2_{k_2,j}) = B^1_{k_1}(s^1_{k_1,j})$;
5) all activation functions are ReLU.

C. Formal Definition of the Merge Function

Here, we provide the formal definition of the *merge* function, which we utilized to construct a combined network to check the (L, ϵ)-conformance between two neural networks. We assume that the two neural networks have the same number of layers, otherwise, we use the append to preprocess them.

Definition 7 (Merge Two Networks): Given two neural networks, $N_1 = (k_1, Act, S^1_i, W^1_i, B^1_i, \sigma^1_i)$ and $N_2 = (k_2, Act, S^2_i, W^2_i, B^2_i, \sigma^2_i)$, with $k_1 = k_2$, $|S^1_0| = |S^2_0|$, $|S^1_{k_1}| = |S^2_{k_2}|$. The merge operation merge(N_1, N_2) returns a new network $N_3 = (k_3, Act, S^3_i, W^3_i, B^3_i, \sigma^3_i)$ where

1) $k_3 = k_1 + 2$;
2) $S^3_0 = S^1_0 \uplus S^2_0$, and $S^3_1 = S'_1 \uplus S^1_1 \uplus S^2_1$, where $S'_1 = \{s'_{1,1}, s'_{1,2}, \ldots, s'_{1,2r_{01}}\}$;
   for each $i \in [2, k_1]$, $S^3_i = S'_i \uplus S^1_i \uplus S^2_i$, where $S'_i = \{s'_i\}$;
   $S^3_{k_1+1} = \{s'_{k_1+1}, s_{k_1+1,1}, s_{k_1+1,1}, \ldots, s_{k_1+1,2r_{k_1 1}}\}$,
   $S^3_{k_1+2} = \{s', s\}$.
3) We define the weight functions as follows. For all the edges that are present in one of the neural networks, the weights on those edges remain the same, for the ones that we do not mention below are given the weight 0

$$W^3_1\left(s^1_{0,i}, s'_{1,i}\right) = 1;\ W^3_1\left(s^2_{0,i}, s'_{1,r_{01}+i}\right) = 1;$$

$$W^3_1\left(s^1_{0,i}, s'_{1,r_{01}+i}\right) = -1;\ W^3_1\left(s^2_{0,i}, s'_{1,i}\right) = -1;$$

$$W^3_{k_1+2}\left(s'_{k_1+1}, s'\right) = 1;\ W^3_{k_1+1}\left(s^1_{k_1,i}, s_{k_1+1,i}\right) = 1;$$

$$W^3_{k_1+1}\left(s^2_{k_1,i}, s_{k_1+1,r_{k_1 1}+i}\right) = 1;$$

$$W^3_{k_1+1}\left(s^1_{k_1,i}, s_{k_1+1,r_{k_1 1}+i}\right) = -1;$$

$$W^3_{k_1+1}\left(s^2_{k_1,i}, s_{k_1+1,i}\right) = -1$$

   for each $i \in (2r_{01}]$, $W^3_2(s'_{1,i}, s'_2) = 1$;
   for each $i \in [3, k_1 + 1]$, $W^3_i(s'_{i-1}, s'_i) = 1$; and
   for each $i \in (2r_{k_1 1}]$, $W^3_{k_1+2}(s_{k_1+1,i}, s) = 1$;
4) We define the bias functions as follows: for all the nodes present in one of the neural networks, their biases remain the same. For all the newly added nodes, the bias value is set to 0.
5) All activation functions are ReLU.

References

[1] B. Paulsen, J. Wang, and C. Wang, "Reludiff: Differential verification of deep neural networks," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng. (ICSE)*, 2020, pp. 714–726.

[2] S. Teuber, M. K. Büning, P. Kern, and C. Sinz, "Geometric path enumeration for equivalence verification of neural networks," in *Proc. IEEE 33rd Int. Conf. Tools Artif. Intell. (ICTAI)*, 2021, pp. 200–208.

[3] C. Eleftheriadis, N. Kekatos, P. Katsaros, and S. Tripakis, "On neural network equivalence checking using SMT solvers," 2022, *arXiv:2203.11629*.

[4] P. Habeeb and P. Prabhakar, "Approximate conformance verification of deep neural networks," in *Proc. 16th Int. Symp. NASA Formal Methods (NFM)*, Moffett Field, CA, USA, 2024, pp. 223–238. [Online]. Available: https://doi.org/10.1007/978-3-031-60698-4_13

[5] S. Bak, H.-D. Tran, K. Hobbs, and T. T. Johnson, "Improved geometric path enumeration for verifying ReLU neural networks," in *Proc. Int. Conf. Comput. Aided Verificat.*, 2020, pp. 66–96.

[6] G. Katz et al., "The marabou framework for verification and analysis of deep neural networks," in *Proc. Int. Conf. Comput. Aided Verificat.*, 2019, pp. 443–452.

[7] S. Wang et al., "Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, 2021, pp. 1–13.

[8] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Proc. 29th Int. Conf., Comput. Aided Verificat. (CAV)*, Heidelberg, Germany, 2017, pp. 97–117. [Online]. Available: https://doi.org/10.1007/978-3-319-63387-9_5

[9] P. Prabhakar and Z. R. Afzal, "Abstraction based output range analysis for neural networks," in *Proc. 32nd Annu. Conf. Neural Inf. Process. Syst. NeurIPS*, Vancouver, BC, Canada, 2019, pp. 15762–15772.

[10] K. Hornik, M. B. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://doi.org/10.1016/0893-6080(89)90020-8

[11] H. Zhang, T. Weng, P. Chen, C. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Proc. 31st Annu. Conf. Neural Inf. Process. Syst. NeurIPS*, Montréal, QC, Canada, 2018, pp. 4944–4953.

[12] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Proc. 15th Int. Symp. Autom. Technol. Verificat. Anal. (ATVA)*, 2017, pp. 269–286. [Online]. Available: https://doi.org/10.1007/978-3-319-68167-2_19

[13] L. Pulina and A. Tacchella, "Never: A tool for artificial neural networks verification," *Ann. Math. Artif. Intell.*, vol. 62, nos. 3–4, pp. 403–425, 2011. [Online]. Available: https://doi.org/10.1007/s10472-011-9243-0

[14] R. Lal and P. Prabhakar, "Abstraction-based safety analysis of linear dynamical systems with neural network controllers," in *Proc. 62nd IEEE Conf. Decision Control*, Singapore, 2023, pp. 8006–8011.

[15] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: safety and robustness certification of neural networks with abstract interpretation," in *Proc. IEEE Symp. Security Privacy, (SP)*, San Francisco, CA, USA, 2018, pp. 3–18. [Online]. Available: https://doi.org/10.1109/SP.2018.00058

[16] S. Dutta, X. Chen, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Sherlock-a tool for verification of neural network feedback systems: Demo abstract," in *Proc. 22nd ACM Int. Conf. Hybrid Syst. Comput. Control, (HSCC)*, Montréal, QC, Canada, 2019, pp. 262–263. [Online]. Available: https://doi.org/10.1145/3302504.3313351

[17] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "An abstract domain for certifying neural networks," *Proc. ACM Program. Lang.*, vol. 3, pp. 1–41, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3290354

[18] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[19] G. Brockman et al., "Openai gym," 2016, *arXiv:1606.01540*.

[20] S. Bak, "nnenum: Verification of ReLU neural networks with optimized abstraction refinement," in *Proc. NASA Formal Methods Symp.*, 2021, pp. 19–36.

[21] D. M. Lopez, M. Althoff, M. Forets, T. T. Johnson, T. Ladner, and C. Schilling, "Arch-comp23 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants," in *Proc. 10th Int. Workshop Appl. Verificat. Continuous Hybrid Syst. (ARCH23)*, 2023, pp. 89–125. [Online]. Available: https://easychair.org/publications/paper/Vfq4b

[22] H. Tran et al., "NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," in *Proc. 32nd Int. Conf. Comput. Aided Verificat. (CAV)*, Los Angeles, CA, USA, 2020, pp. 3–17. [Online]. Available: https://doi.org/10.1007/978-3-030-53288-8_1

[23] Y. Zhou and S. Tripakis, "Compositional inductive invariant based verification of neural network controlled systems," in *Proc. 16th Int. Symp., NASA Formal Methods (NFM)*, Moffett Field, CA, USA, 2024, pp. 239–255. [Online]. Available: https://doi.org/10.1007/978-3-031-60698-4_14

[24] F. Rossi, C. Bernardeschi, M. Cococcioni, and M. Palmieri, "Towards formal verification of neural networks in cyber-physical systems," in *Proc. 16th Int. Symp. NASA Formal Methods (NFM)*, Moffett Field, CA, USA, 2024, pp. 207–222. [Online]. Available: https://doi.org/10.1007/978-3-031-60698-4_12

[25] X. Sun, H. Khedr, and Y. Shoukry, "Formal verification of neural network controlled autonomous systems," in *Proc. 22nd ACM Int. Conf. Hybrid Syst. Comput. Control, (HSCC)*, Montreal, QC, Canada, 2019, pp. 147–156. [Online]. Available: https://doi.org/10.1145/3302504.3311802

[26] R. Milner, *English Communication and Concurrency*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[27] A. Girard and G. J. Pappas, "Approximation metrics for discrete and continuous systems," *IEEE Trans. Autom. Control.*, vol. 52, no. 5, pp. 782–798, May 2007. [Online]. Available: https://doi.org/10.1109/TAC.2007.895849

[28] A. A. Julius, A. D'Innocenzo, M. D. D. Benedetto, and G. J. Pappas, "Approximate equivalence and synchronization of metric transition systems," *Syst. Control. Lett.*, vol. 58, no. 2, pp. 94–101, 2009. [Online]. Available: https://doi.org/10.1016/j.sysconle.2008.09.001

[29] M. G. Soto and P. Prabhakar, "Hybridization for stability verification of nonlinear switched systems," in *Proc. 41st IEEE Real-Time Syst. Symp., (RTSS)*, Houston, TX, USA, 2020, pp. 244–256. [Online]. Available: https://doi.org/10.1109/RTSS49844.2020.00031

[30] H. Roehm, J. Oehlerking, M. Woehrle, and M. Althoff, "Reachset conformance testing of hybrid automata," in *Proc. 19th Int. Conf. Hybrid Syst., Comput. Control, (HSCC)*, Vienna, Austria, 2016, pp. 277–286. [Online]. Available: https://doi.org/10.1145/2883817.2883828

[31] P. Prabhakar, "Bisimulations for neural network reduction," in *Proc. 23rd Int. Conf. Verificat., Model Check., Abstract Interpretat. (VMCAI)*, Philadelphia, PA, USA, 2022, pp. 285–300. [Online]. Available: https://doi.org/10.1007/978-3-030-94583-1_14