SYNERGISTIC VERIFICATION OF NEURAL NETWORK CONTROLLED SYSTEMS VIA ADVERSARIALLY GUIDED REFINEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural network controlled systems (NNCSs) are increasingly deployed in safety-critical applications, making the safety property verification a fundamental challenge. Static reachability-based analysis offers formal safety guarantees but often suffers from overapproximation errors, leading to conservative or "Unknown" results, while dynamic falsification methods can efficiently discover counterexamples but provide no guarantees in their absence. In this paper, we propose a novel synergistic verification framework via adversarially guided refinement, which integrates dynamic falsification with static verification in a unified loop. The core idea is to integrate backward reachability analysis with projected gradient descent (PGD) based adversarial testing to efficiently explore safe behaviors, along with exploiting adversarial trajectories to guide targeted branch-and-bound refinement over critical time steps and neurons. Experimental results demonstrate our approach significantly enhances verification effectiveness, achieving up to 85% speedups, while producing tighter reachable set approximations and more conclusive verification outcomes for complex NNCSs.

1 INTRODUCTION

Neural network controlled systems (NNCSs) have become prevalent in autonomous driving, robotics, et al. Zhang et al. (2019); Hagan et al. (2002); Hunt et al. (1992). Due to the complex interactions between neural controllers and dynamical environments, ensuring their safety remains nontrivial. It aims to ensure that, for all admissible initial states, the evolving trajectories remain within the safe region and avoid a predefined unsafe set over the time horizon of interest. To establish such guarantees, both static verification and dynamic falsification have been widely adopted. Static verification provides sound guarantees on safety properties, whereas dynamic falsification enables the efficient discovery of concrete failure behaviors.

From the perspective of static verification, reachability-based methods have been extensively studied through forward and backward set propagation. Forward approaches employ abstraction domains such as Starsets and Taylor models to overapproximate closed-loop behaviors, as exemplified by *NNV2.0* Lopez et al. (2023) and *CROWN-Reach* Manzanas Lopez et al. (2024), while backward approaches, including *BReach-LP* Rober et al. (2022) and *DRIP* Everett et al. (2023), compute predecessor states from unsafe regions to focus verification on safety-critical areas. Nevertheless, forward analysis suffers from accumulated overapproximation errors, whereas backward analysis faces scalability challenges over long horizons, and neither can efficiently expose concrete unsafe behaviors. Dynamic falsification addresses this limitation through system-level exploration, where recent methods exploit temporal coverage, execution traces, and neuron activation patterns to discover safety-critical behaviors Zhang et al. (2022); Lyu et al. (2025b;a). However, these often explore high-dimensional state spaces inefficiently, lack sufficient focus on adversarially sensitive regions, and cannot provide formal safety guarantees in the absence of counterexamples.

To bridge the gap, we propose a synergistic framework via adversarially guided refinement for effective and rigorous safety verification of NNCSs. The proposed framework tightly integrates dynamic adversarial testing with static reachability analysis in an iterative manner, where adversarial trajectories are exploited to guide the refinement of safety-critical regions. In particular, potential unsafe

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

behaviors are leveraged to identify critical time steps and neurons for targeted branch-and-bound refinement, thereby reducing overapproximation errors during reachable set propagation. By systematically incorporating adversarial information into formal analysis, the proposed approach enables more focused exploration of critical behaviors, mitigates conservativeness in reachability analysis, and improves the precision and decisiveness of verification outcomes. The main contributions of this work are summarized as follows:

- We propose a synergistic verification framework that integrates dynamic falsification with static verification, enabling resolve inconclusive *"Unknown"* verification outcomes of the neural network controlled systems.
- We leverage an adversarially guided refinement strategy that exploits trajectory-level information from dynamic falsification to identify critical time steps and neurons, enabling targeted branch-and-bound refinement of static reachability analysis.
- We develop a verification tool *SynVAR* and achieve up to 85% efficiency improvement across representative benchmarks over state-of-the-art approaches.

2 PRELIMINARIES

2.1 PROBLEM STATEMENT

We consider a neural network controlled system, which modeled as the following discrete-time dynamical system:

$$x_{t+1} = f(x_t, u_t), \qquad u_t = n(x_t), \qquad t \geq 0, \tag{1}$$

where $x_t \in \mathbb{R}^{n_x}$ denotes the system state, $u_t \in \mathbb{R}^{n_u}$ the control input, f the plant dynamics, and n the NN controller. Starting from an initial state $x_0 \in \mathcal{X}_0$, the closed-loop system evolves according to (1), generating a state trajectory $\{x_0, x_1, \ldots, x_t\}$. The controller $n(\cdot)$ is modeled as an L-layer feedforward neural network

$$n(x) = n_L \circ n_{L-1} \circ \cdots \circ n_1(x), \quad n_i : x \mapsto \sigma(W_i x + b_i),$$

with W_i and b_i denoting the weight matrix and bias vector of layer i, and σ the activation function.

Problem. Given an initial state set and sampling period τ, at each step k, the controller reads x_k and outputs $u_k = \sigma(x_k)$, and the plant evolves to x_{k+1} under this input. Our goal is to check all possible initial states and closed-loop trajectories within a given time horizon to ensure compliance with the safety specifications formally defined as follows.

Definition 1 (Safety Property) *Given an initial state set $\mathcal{X}_0$, let $\mathcal{R}_t$ denote the reachable set of the closed-loop NNCS at time step t. For an unsafe region $\mathcal{X}_{\text{unsafe}} \subseteq \mathbb{R}^{n_x}$ and a finite horizon T, the NNCS satisfies the safety property if $\mathcal{R}_t \cap \mathcal{X}_{\text{unsafe}} = \emptyset, \forall\, t \in \{0, \ldots, T\}$.*

2.2 STATIC VERIFICATION VIA REACHABILITY ANALYSIS

Static verification of NNCSs is typically performed via reachability analysis, which computes overapproximations of reachable sets under closed-loop dynamics.

[Forward and Backward Reachability]. Forward reachability analysis propagates reachable sets from the initial state set $\mathcal{X}_0$ over time. At each time step t, the reachable set is computed as

$$\mathcal{U}_t = \kappa(n(\mathcal{X}_t)), \qquad \mathcal{R}_f^{(t+1)} = \tau(f(\mathcal{X}_t, \mathcal{U}_t)), \qquad \mathcal{X}_{t+1} = p(\mathcal{R}_f^{(t+1)}),$$

where $\kappa(\cdot)$ and $\tau(\cdot)$ compute sound overapproximations of the neural network output and plant dynamics, respectively, and $p(\cdot)$ projects the resulting set onto the state space.

Backward reachability analysis starts from unsafe region $\mathcal{X}_{\text{unsafe}}$ and recursively computes its preimages. Let $\mathcal{R}_b^{(0)} = \mathcal{X}_{\text{unsafe}}$. For $k = 1, \ldots, T$, the k-step backward reachable set is computed as

$$\mathcal{R}_b^{(k)} = \left\{ x \in \mathbb{R}^{n_x} \;\middle|\; \exists u \in \mathcal{U} \text{ s.t. } f(x,u) \in \mathcal{R}_b^{(k-1)} \right\}.$$

The cumulative backward reachable set over the verification horizon T is then $\mathcal{R}_b^{[0:T]} = \bigcup_{k=0}^{T} \mathcal{R}_b^{(k)}$,which contains the states that can reach the unsafe region within at most T time steps.

[Symbolic Interval Abstraction and Linear Relaxation]. To enable tractable set propagation through neural network controllers, overapproximation techniques such as symbolic interval abstraction are employed. By combining symbolic bound propagation with linear relaxation, sound affine bounds on network outputs can be obtained for reachability analysis.

Given an input set $\mathcal{X}$, intermediate variables are represented by affine lower and upper bounds with respect to the input

$$\underline{v}(x) = A_v^{(l)} x + b_v^{(l)}, \quad \overline{v}(x) = A_v^{(u)} x + b_v^{(u)},$$

such that $\underline{v}(x) \le v \le \overline{v}(x)$ for all $x \in \mathcal{X}$. Consider a feedforward neural network with

$$z^{(k)} = W^{(k)} a^{(k-1)} + b^{(k)}, \quad a^{(k)} = \sigma(z^{(k)}), \quad a^{(0)} = x.$$

For linear layers, bounds are propagated via weight decomposition as follows:

$\overline{z}^{(k)}(x) = W_+^{(k)} \overline{a}^{(k-1)}(x) + W_-^{(k)} \underline{a}^{(k-1)}(x) + b^{(k)}$, $\underline{z}^{(k)}(x) = W_+^{(k)} \underline{a}^{(k-1)}(x) + W_-^{(k)} \overline{a}^{(k-1)}(x) + b^{(k)}$.

where $W_+ = \max(W, 0)$ and $W_- = \min(W, 0)$.

For nonlinear activations such as ReLU, convex linear relaxation is applied. Given $x \in [\ell, u]$, the relaxation is

$$y \ge 0, \quad y \ge x, \quad y \le \lambda x + \mu,$$

with $\lambda = \frac{u}{u-\ell}$, $\mu = -\frac{u\ell}{u-\ell}$. Thus, affine bound propagation enables sound reachable-set overapproximation by expressing network outputs as linear input functions.

3 Main Methods

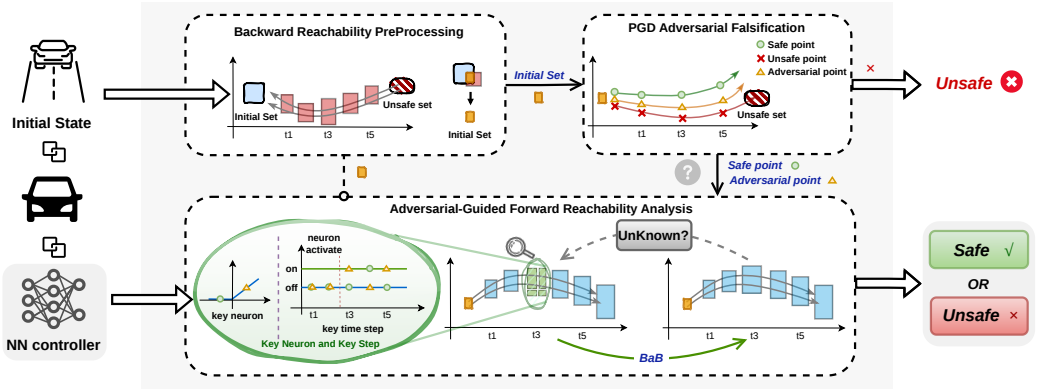


Figure 1. The framework of synergistic verification for NNCSs.

We propose a framework named *SynVAR*, as illustrated in Fig. 1, which integrates dynamic falsification with static verification through an iterative verification-refinement procedure, consisting of three main components: **(i) backward reachability preprocessing**, which derives time-indexed safety constraints to refine the initial state set and reduce the subsequent search space; **(ii) PGD adversarial falsification**, which explores potential unsafe behaviors and identifies critical time steps and neurons based on activation discrepancies; and **(iii) adversarial-guided forward reachability verification**, which performs reachable set propagation and selectively applies branch-and-bound (BaB) refinement on critical neurons and time steps to reduce overapproximation errors when verification remains inconclusive (*"Unknown"*).

The overall procedure is summarized in Algorithm 1. Backward reachability (Lines 1–2) first reconstructs a refined initial set $\widetilde{\mathcal{X}}_0$. PGD-based adversarial falsification (Line 3) then searches for unsafe trajectories; if a counterexample ((i.e., a concrete trajectory that reaches the unsafe set)) is found, the algorithm terminates immediately with *"Unsafe"*(Lines 4–6). Otherwise, an iterative verification-refinement loop (Lines 8–14) alternates between forward reachability analysis and BaB refinement until a conclusive result is obtained or the refinement budget is exhausted.

Algorithm 1 Synergistic **V**erification of NNCSs via **A**dversarially Guided **R**efinement (***SynVAR***)

Input: NN controller nn, system dynamics f, initial state set $\mathcal{X}_0$, unsafe set $\mathcal{X}_{\text{unsafe}}$, time horizon T, BaB budget B
Output: result $\mathcal{S} \in \{\textit{Safe}\checkmark, \textit{Unsafe}\times, \textit{Unknown}\}$

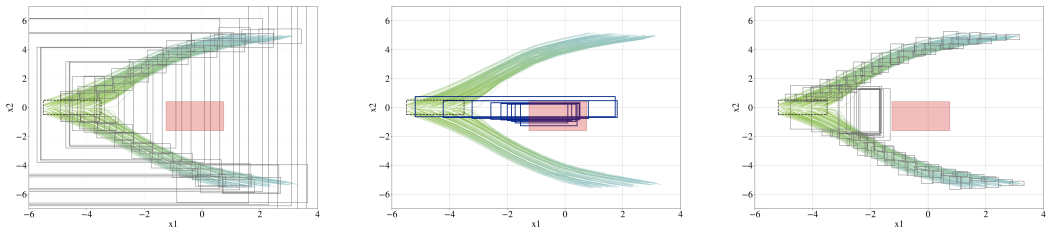
$\{\mathcal{C}_t\}_{t=0}^{T} \leftarrow \text{BackwardReach}(\mathcal{X}_{\text{unsafe}})$
$\widetilde{\mathcal{X}}_0 \leftarrow \mathcal{X}_0 \cap \mathcal{C}_T$
$(\mathcal{P}_{\text{safe}}, \mathcal{P}_{\text{pot}}, \mathcal{P}_{\text{unsafe}}) \leftarrow \text{PGD}(\widetilde{\mathcal{X}}_0, nn, f, \mathcal{X}_{\text{unsafe}}, T, B)$
▷ reference to Algorithm 2
if $\mathcal{P}_{\text{unsafe}} \neq \emptyset$ **then**
  return *Unsafe*✗
end if
$\mathcal{S} \leftarrow \textit{Unknown}$
while $\mathcal{S} = \textit{Unknown}$ **do**
  $\mathcal{S} \leftarrow \text{ForwardReachAndCheck}(nn, f, \mathcal{X}_0, \mathcal{X}_{\text{unsafe}}, T)$
  ▷ reference to Algorithm 3
  if $\mathcal{S} \neq \textit{Unknown}$ **then**
    return $\mathcal{S}$
  end if
  $\mathcal{S} \leftarrow \text{BaBRefinement}(\mathcal{P}_{\text{safe}}, \mathcal{P}_{\text{pot}})$
end while
return $\mathcal{S}$

3.1 Backward Reachability Preprocessing

To alleviate the accumulation of overapproximation errors in forward reachability analysis, we introduce a backward reachability preprocessing stage to identify safety-critical regions of the state space. Starting from the unsafe set $\mathcal{X}_{\text{unsafe}}$, we iteratively compute overapproximations of predecessor sets over a finite time horizon, yielding a sequence of time-indexed constraint sets $\{\mathcal{C}_t\}_{t=0}^{T}$. Each $\mathcal{C}_t$ characterizes the set of states at time step t that may evolve into unsafe set within the remaining horizon. Based on the backward reachable set at the initial time, we refine the initial set as

$$\widetilde{\mathcal{X}}_0 = \mathcal{X}_0 \cap \mathcal{C}_T,$$

thereby restricting subsequent analysis to safety-critical regions. This refinement reduces the volume of reachable sets to be propagated and mitigates the accumulation of overapproximation errors in forward analysis.



(a) Normal forward reachability (b) Backward reachability (c) Refined forward reachability

Figure 2. Motivating example of backward reachability preprocessing on the *Ground Robot* Rober et al. (2022). The red region indicates the unsafe set, and green curves correspond to system trajectories. Gray rectangles denote forward reachable sets for illustrative purposes, and blue rectangles represent backward reachable sets computed during preprocessing.

Example 1. We illustrate the effect of backward reachability preprocessing using the *Ground Robot* benchmark Rober et al. (2022). The system is modeled as a 2D discrete-time integrator:

$$x_{t+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u_t.$$

The initial state space is defined as $[-5.5, -3.5] \times [-0.5, 0.5]$, and the unsafe region is given by $[-1.25, 0.75] \times [-1.6, 0.4]$, with a sampling time of $t_s = 1\,\text{s}$.

As shown in Fig. 2a, conventional BaB-based forward methods struggle to partition the system states into the upper and lower regions. Consequently, some reachable sets simultaneously contain states from both trajectories at each time step, causing the reachable sets intersect with the unsafe region and making it difficult to verify the system safety properties. Fig. 2b illustrates how backward reachability analysis reconstructs the system states, while Fig. 2c shows the BaB method combined with backward analysis can successfully verify the safety properties. The figure shows how backward preprocessing focuses verification on safety-critical regions and reduces unnecessary exploration.

3.2 PGD-BASED ADVERSARIAL ATTACK FOR NNCSS

In this section, we present a gradient-based adversarial falsification method for NNCSs. Instead of relying on sampling-based black-box exploration, we exploit the differentiability of the closed-loop dynamics to optimize the initial state via trajectory-level gradients, thereby efficiently searching for potential safety violations.

To characterize trajectory evolution, let $F_k : \mathbb{R}^n \to \mathbb{R}^n$ denote the k-step state mapping, i.e., $x_k = F_k(x_0)$. We measure proximity to the unsafe region $\mathcal{X}_{\text{unsafe}}$ using the signed distance $d(x, \mathcal{X}_{\text{unsafe}})$, defined as $d(x, \mathcal{X}_{\text{unsafe}}) = -\inf_{y \in \mathcal{X}_{\text{unsafe}}} \|x - y\|_2$ for $x \notin \mathcal{X}_{\text{unsafe}}$ and $d(x, \mathcal{X}_{\text{unsafe}}) = \inf_{y \notin \mathcal{X}_{\text{unsafe}}} \|x - y\|_2$ otherwise. The trajectory risk over horizon N is then

$$\mathcal{R}(x_0) = \max_{k=0,\dots,N} d(F_k(x_0), \mathcal{X}_{\text{unsafe}}), \tag{2}$$

whose maximization drives the trajectory toward or into the unsafe region. Directly evaluating the signed distance can be computationally expensive. For $\mathcal{X}_{\text{unsafe}} = \{x \mid g_i(x) \le 0,\ i = 1, \dots, s\}$, we therefore use the constraint-violation surrogate $V(x) = -\max_{i \in \{1,\dots,s\}} g_i(x)$ and optimize the trajectory-level loss

$$J_V(x_0) = \max_{k \in \{0,\dots,N\}} V(F_k(x_0)). \tag{3}$$

The initial state is constrained to the hyper-rectangle $\mathcal{X}_0 = \{x \mid l_i \le x[i] \le u_i,\ i = 1, \dots, n\}$ and optimized using PGD through the following four steps:

- **Step 1: Trajectory-level objective construction.** The optimization variable is the initial state x_0 rather than the network input. The objective is to maximize the trajectory-level loss (3), which drives the closed-loop trajectory toward unsafe region over the entire horizon.
- **Step 2: Forward simulation and safety checking.** Given current initial state, the closed-loop system is simulated to obtain $\mathcal{P}^{(t)} = \{x_k^{(t)}\}_{k=0}^{N}$. If any state satisfies unsafe constraints, a genuine counterexample is found and verification immediately returns *Unsafe*.
- **Step 3: Gradient-based PGD update.** Otherwise, the trajectory loss is differentiated with respect to the initial state via backpropagation through time (BPTT Werbos (2002)), yielding $g^{(t)} = \nabla_{x_0} J_V(x_0)\big|_{x_0 = x_0^{(t)}}$. The initial state is then updated by PGD:

  $$x_0^{(t+\frac{1}{2})} = x_0^{(t)} + \alpha \cdot \text{sign}(g^{(t)}), \qquad x_0^{(t+1)} = \text{Clip}_{\mathcal{X}_0}(x_0^{(t+\frac{1}{2})}),$$

  where $\text{Clip}_{\mathcal{X}_0}(x)[i] = \min(\max(x[i], l_i), u_i)$, ensuring that every iterate remains within the feasible initial set.
- **Step 4: Multiple restarts and trajectory selection.** To alleviate local optima, PGD is executed from multiple initializations. If no counterexample is found, the trajectory with the largest objective value is selected as the potentially unsafe trajectory $\mathcal{P}_{\text{pot}}$, while the trajectory farthest from the unsafe region is selected as the representative safe trajectory $\mathcal{P}_{\text{safe}}$ for subsequent BaB refinement.

Algorithm 2 presents the PGD-based adversarial attack procedure for NNCSs. During the R restart iterations, the algorithm first samples an initial state (Line 3). Then, for each of the K PGD iterations, forward simulation is performed to obtain a trajectory $\mathcal{P}$. If the trajectory enters the unsafe region, an unsafe trajectory $\mathcal{P}_{\text{unsafe}}$ is identified (Lines 6-9). Otherwise, the trajectory is used to update the best safe $\mathcal{P}_{\text{safe}}$ or adversarial trajectories $\mathcal{P}_{\text{pot}}$ (Line 10). Then, a PGD update is performed (Line 11). Upon termination, the algorithm returns the above three categories of trajectories.

Algorithm 2 PGD-based adversarial attack for NNCSs

Input: System dynamics f, NN controller nn, initial set $\mathcal{X}_0$, unsafe set $\mathcal{X}_{\text{unsafe}}$, time horizon N, number of random restarts R, number of PGD iterations K, step size α
Output: $\mathcal{P}_{\text{safe}},\ \mathcal{P}_{\text{pot}},\ \mathcal{P}_{\text{unsafe}}$

```
Initialize P_safe, P_pot, P_unsafe
for r = 1 to R do
    x_0 ← Sample(X_0)
    for k = 1 to K do
        P ← ForwardSimulate(f, nn, x_0, N)
        if IsUnsafe(P, X_unsafe) then
            P_unsafe ← P_unsafe ∪ P
            return P_unsafe
        end if
        (P_safe, P_pot) ← UpdateBestTraj(P, P_safe, P_pot)
        x_0 ← PGDStep(x_0, f, nn, X_unsafe, X_0, N, α)
    end for
end for
return (P_safe, P_pot, P_unsafe)
```

3.3 Adversarially Guided BaB Refinement

When Algorithm 1 returns *"Unknown"*, dynamic phase has produced representative safe and potentially unsafe trajectories $(\mathcal{P}_{\text{safe}}, \mathcal{P}_{\text{pot}})$. We exploit resulting activation discrepancies to identify critical time steps and key neurons, guiding BaB refinement toward the most uncertain ReLU branches. Resulting branch constraints are then incorporated into symbolic interval propagation to tighten reachable-set bounds.

Consider a closed-loop execution over the horizon $t = 0, \dots, T$ with a safe trajectory $\mathcal{P}_{\text{safe}} = \{x_t^S\}_{t=0}^T$ and a potentially unsafe trajectory $\mathcal{P}_{\text{pot}} = \{x_t^P\}_{t=0}^T$. Let $\hat{z}_j^{(l),S}(t)$ and $\hat{z}_j^{(l),P}(t)$ denote the corresponding pre-activation values of neuron (l, j) along the two trajectories. Their activation discrepancy is defined as $d_{l,j}(t) = \left\| \hat{z}_j^{(l),P}(t) - \hat{z}_j^{(l),S}(t) \right\|_2$. To capture temporal influence, we accumulate discrepancies using

$$D(t) = \frac{1}{\sum_{l=1}^{L} n_l} \sum_{l=1}^{L} \sum_{j=1}^{n_l} d_{l,j}(t) + \lambda D(t-1), \qquad D(-1) = 0,$$

where $\lambda \in [0, 1)$ is the decay factor. The critical time steps and neurons are selected according to their importance scores as

$$\mathcal{T}^\star = \text{TopK}\big(\{D(t)\}_{t=0}^T, K_T\big), \quad \mathcal{N}^\star(t) = \text{TopK}(\{d_{l,j}(t)\}, K_N), \quad t \in \mathcal{T}^\star.$$

BaB verification is then performed only on these critical neurons. Each BaB node is represented as $\mathcal{N} = (\mathcal{X}_0^{\mathcal{N}}, \mathcal{A}^{\mathcal{N}}, \mathcal{I}^{\mathcal{N}})$, where $\mathcal{X}_0^{\mathcal{N}} \subseteq \mathcal{X}_0$ is the initial-state subset, and $\mathcal{A}^{\mathcal{N}}$ and $\mathcal{I}^{\mathcal{N}}$ denote neurons fixed as active and inactive, respectively. The corresponding reachable set is $\mathcal{R}_t^{\mathcal{N}} = \text{Reach}(\mathcal{X}_0^{\mathcal{N}}, \mathcal{A}^{\mathcal{N}}, \mathcal{I}^{\mathcal{N}}, t)$. Only unstable ReLU neurons satisfying $\hat{\underline{z}}_j^{(l)}(t) < 0 < \hat{\overline{z}}_j^{(l)}(t)$ are split. For a selected neuron $(t^\star, l^\star, j^\star)$, binary branching produces

$$\mathcal{N}^+ : \hat{z}_{j^\star}^{(l^\star)}(t^\star) > 0, \qquad \mathcal{N}^- : \hat{z}_{j^\star}^{(l^\star)}(t^\star) \le 0,$$

which satisfy $\mathcal{R}_t^{\mathcal{N}} = \mathcal{R}_t^{\mathcal{N}^+} \cup \mathcal{R}_t^{\mathcal{N}^-}$. Branch constraints are injected into symbolic interval propagation to tighten reachable set bounds. For affine layers,

$$\hat{\underline{z}} = W_+ l + W_- u + b, \qquad \hat{\overline{z}} = W_+ u + W_- l + b,$$

where $W_+ = \max(W, 0)$ and $W_- = \min(W, 0)$. The ReLU output interval is updated as $[\max(\hat{\underline{z}}, 0), \max(\hat{\overline{z}}, 0)]$, while infeasible branches are pruned whenever the imposed branch constraints contradict the interval bounds. The recursive verification procedure is

$$\text{Verify}(\mathcal{N}, B) = \begin{cases} s, & s \neq \textit{Unknown} \ \vee \ B = 0, \\ \text{Agg}(\mathcal{N}, B-1), & \text{otherwise}, \end{cases}$$

where $s = \mathrm{Status}(\mathcal{N})$. The function $\mathrm{Agg}(\mathcal{N}, B-1)$ aggregates the verification results of its child branches. The aggregation function returns *Unsafe* if any child branch is unsafe, *Safe* if all child branches are safe, and *Unknown* otherwise. By combining adversarial trajectory guidance with BaB refinement, the search focuses on the regions responsible for the largest overapproximation precision loss, substantially reducing the search space while improving verification efficiency and precision.

Algorithm 3 Adversarial-Sample-Guided BaB Refinement

Input: Neural network controller nn, system dynamics f, reconstructed initial set $\widetilde{\mathcal{X}}_0$, unsafe set $\mathcal{X}_{\text{unsafe}}$, time horizon T, trajectory sets $(\mathcal{P}_{\text{safe}}, \mathcal{P}_{\text{pot}})$, BaB budget B
Output: Verification result $\mathcal{S} \in \{$*Safe*✓, *Unsafe*✗$\}$
1: $\mathcal{S} \leftarrow$ *Unknown*
2: $\{D(t)\}_{t=0}^{T}$, $\{d_{l,j}(t)\}_{t,l,j} \leftarrow$ Discrepancy$(\mathcal{P}_{\text{safe}}, \mathcal{P}_{\text{pot}})$
3: $(\mathcal{T}^\star, \{\mathcal{N}^\star(t)\}_{t\in\mathcal{T}^\star}) \leftarrow$ Critical$(\{D(t)\}, \{d_{l,j}(t)\})$
4: **while** $\mathcal{S} =$ *Unknown* **and** $B > 0$ **do**
5: $\quad t^\star \leftarrow$ SelectTimeStep$(\mathcal{T}^\star)$
6: $\quad (l^\star, j^\star) \leftarrow$ SelectNeuron$(\mathcal{N}^\star(t^\star))$
7: $\quad \Gamma \leftarrow$ SplitNeuron$(t^\star, l^\star, j^\star)$
8: $\quad \mathcal{S} \leftarrow$ BaBVerify$(nn, f, \widetilde{\mathcal{X}}_0, \Gamma, \mathcal{X}_{\text{unsafe}}, T)$
9: $\quad B \leftarrow B - 1$
10: **end while**
11: **return** $\mathcal{S}$

Algorithm 3 summarizes the refinement procedure. The algorithm first computes the activation discrepancy scores, denoted by *Discrepancy*$(\cdot)$, from the safe and adversarial trajectories (Line 2). It then invokes *Critical*$(\cdot)$ to identify the critical time steps and neurons for subsequent refinement (Line 3). The algorithm then backtracks from the highest-ranked time step and iteratively selects critical neurons for refinement (Lines 5–6). For each selected neuron, *SplitNeuron*$(\cdot)$ generates two child branches by fixing its ReLU activation status, and *BaBVerify*$(\cdot)$ recursively verifies the resulting subproblems (Lines 7–8).

4 Experiments

In this section, we conduct a series of experiments to evaluate our tool *SynVAR*. We first present representative benchmarks as case studies to illustrate the workflow and behavior of *SynVAR*. We then conduct a comparison against SOTA verification tools from the ARCH-COMP25 Lopez et al. (2025) competition. The evaluation focuses on two aspects: (i) verification effectiveness, measured by the ability to produce conclusive results (*"Safe"* or *"Unsafe"*) as opposed to *"Unknown"* (i.e., no definitive verification conclusion can be reached within the given time budget) , and (ii) computational efficiency, measured in terms of runtime. Furthermore, we perform ablation studies to assess the contribution of key components. All experiments are conducted on a machine equipped with an Intel Core i7-12700 CPU (12 cores, 20 threads, up to 4.9 GHz) and 32 GB RAM. The codes of *SynVAR* and implementation details are available in the supplementary materials.

4.1 Case Study

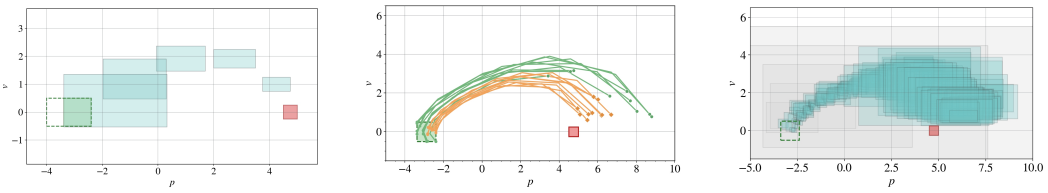
We first present a 2-dimensional benchmark to illustrate the overall procedure in Algorithm 1. We further evaluate *SynVAR* on a more complex 6-dimensional system to demonstrate its scalability, with detailed results provided in Appendix A.4.2.

Example 1 (Double Integrator Hu et al. (2020)) *The system dynamic is*

$$x_{k+1} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} \frac{1}{2}dt^2 \\ dt \end{bmatrix} u_k,$$

where the state x_k consists of position and velocity. The initial state set is given by $[-4, -2.4] \times [-0.5, 0.5]$, with a sampling time of $t_s = 1$ s. The objective is to verify that, over a time horizon of 5 steps, system trajectories remain within safe region and avoid the unsafe set $[4.5, 5] \times [-0.25, 0.25]$.

SynVAR first performs backward reachability preprocessing to exclude states that cannot reach unsafe region, refining the initial set from $[-4, -2.4] \times [-0.5, 0.5]$ to $[-3.38, -2.4] \times [-0.5, 0.5]$ (Fig. 3a). PGD-based adversarial falsification is then conducted over the refined set. Although no counterexample is found, the resulting safe and near-unsafe trajectories (Fig. 3b) guide the subsequent verification. Finally, adversarial-guided BaB refinement tightens forward reachable sets by partitioning critical neurons and time steps identified from activation discrepancies. As shown in Fig. 3c, the refined reachable sets (blue) eliminate the spurious intersection of the baseline reachable sets (gray) with unsafe region at time step 4, successfully verifying safety with only 38 branches.



(a) Backward preprocessing (b) PGD adversarial falsification (c) Reachability analysis

Figure 3. Synergistic verification process of Example 1 using *SynVAR*. The red box denotes the unsafe region, while the green dashed box represents the initial region and the green rectangles indicate the reconstructed initial regions. The green trajectories represent safe sample trajectories, while the orange trajectories represent potentially unsafe sample trajectories yield from PGD. The blue rectangles correspond to the forward/backward reachable sets computed by *SynVAR*, whereas the gray rectangles denote the reachable sets generated by the baseline forward method.

4.2 Performance Evaluation

To comprehensively evaluate performance of *SynVAR*, we investigate following research questions:

RQ1: What is the impact of backward preprocessing and adversarially guided BaB in *SynVAR*?
RQ2: How does *SynVAR* compare with existing SOTA verification tools?

4.2.1 Ablation Study of Key Components

To answer RQ1, we perform an ablation study by excluding backward reachability preprocessing and focusing on the contributions of PGD-based falsification and adversarial-guided BaB refinement. Table 1 compares *SynVAR* with existing tools and indicates whether backward reachability and BaB refinement are employed. A check mark (✓) and a cross (✗) denote safe and unsafe verification results, respectively.

Table 1. Verification results of SynVAR and baselines on four benchmarks

Method	Backward & BaB	Ground Robot		GR Buggy		Double Integrator		Quadrotor	
		Result	Time	Result	Time	Result	Time	Result	Time
SynVAR*	w/ & (w/ *PGD-BaB*)	✓	**1.04s**	✗	**0.16s**	✓	**0.63s**	✓	3.81s
Reach-LP-GSG	w/o & (w/ *GreedySim-BaB*)	*Unknown*	0.39s	*Unknown*	0.38s	*Unknown*	0.39s	*Unknown*	0.57s
ReReach-LP	w/o & w/o	*Unknown*	4.04s	✗	2.80s	✓	1.30s	✓	6.56s
ReachMM-CG	w/o & (w/ *Contraction-BaB*)	*Unknown*	0.36s	✗	1.05s	*Unknown*	0.20s	*Unknown*	0.26s
ReachLibBaB	w/o & (w/ *LipschitzBounds-BaB*)	*Unknown*	34.36s	✗	171.58s	✓	0.65s	*Unknown*	*Timeout*
BReach-LP	w/ & w/o	✓	1.13s	*Unknown*	2.63s	*Unknown*	4.69s	✓	1.89s
DRIP-Hpoly	w/ & w/o	*Unknown*	10.27s	*Unknown*	7.32s	*Unknown*	8.04s	*Unknown*	7.84s
HyBReach-LP	w/ & (w/ *Uniform-BaB*)	✓	2.53s	*Unknown*	20.57s	✓	2.35s	✓	**1.36s**

As shown in Table 1, *SynVAR* successfully verifies all four benchmarks and achieves the best verification efficiency on three of them, reducing verification time by up to 85% over competing methods. On the Ground Robot buggy (GR Buggy) benchmark, *SynVAR* quickly falsifies the safety property. For the high-dimensional Quadrotor benchmark, backward methods (BReach-LP and HyBReach-LP) remain competitive, while *SynVAR* significantly outperforms existing forward reachability approaches. We further evaluate its applicability to longer-horizon verification and real-time safety assurance, with detailed results provided in Appendix A.4.4.

4.2.2 Comparison with State-of-the-Art Verification Methods

To answer RQ2, we evaluate *SynVAR* on widely adopted benchmarks from the ARCH-COMP25 competition Lopez et al. (2025) and compare it against state-of-the-art verification tools participating in the same benchmark suite. The overall results are summarized in Table 2.

Table 2. Verification results of SynVAR and baselines on additional benchmarks

Method	Airplane		Double Pendulum (Robust)		ACC		Single Pendulum	
	Result	Time	Result	Time	Result	Time	Result	Time
SynVAR*	✗	**1.67s**	✗	**0.70s**	✓	0.72s	✓	**0.58s**
CROWN-Reach	✗	5.56s	✗	1.23s	✓	2.53s	✓	0.63s
CORA	✗	2.71s	✗	5.22s	✓	3.09s	✓	3.81s
NNV2.0	✗	8.27s	*Unknown*	20.10s	✓	26.75s	✓	99.48s
JuliaReach	✗	4.44s	✗	0.83s	✓	**0.64s**	✓	11.13s

As shown in Table 2, *SynVAR* achieves the best verification efficiency on three of the four benchmarks, reducing verification time by up to 38% over the strongest baselines. It also detects unsafe behaviors with lower verification cost; for example, on the Double Pendulum benchmark, *NNV2.0* returns *"Unknown"*, whereas *SynVAR* successfully identifies a counterexample. These results demonstrate the effectiveness of integrating backward reachability, PGD-based adversarial falsification, and BaB refinement to focus verification on high-risk regions. In contrast, existing methods without targeted guidance often require more samples or partitions, particularly when unsafe states are sparse or near the input boundary, increasing verification time and reducing falsification efficiency.

5 Related Work

Reachability-based verification has been extensively studied for NNCSs through forward and backward set propagation. Existing methods employ abstraction domains such as zonotopes, Starsets, and Taylor models to overapproximate closed-loop behaviors Bogomolov et al. (2019); Tran et al. (2020); Ladner & Althoff (2024); Althoff; Kochdumper et al. (2023); Manzanas Lopez et al. (2024); Zhang et al. (2018). To improve precision, branch-and-bound refinement and backward reachability have been developed to iteratively partition the state space or compute unsafe preimages Everett et al. (2020); Entesari et al. (2023); Jafarpour et al. (2023); Harapanahalli et al. (2023); Entesari & Fazlyab (2023); Rober et al. (2023b; 2022; 2023a); Everett et al. (2023); Kotha et al. (2023). Nevertheless, these methods remain limited by accumulated overapproximation errors and scalability challenges for complex nonlinear NNCSs.

Complementary to static verification, dynamic falsification explores counterexamples through simulation-based optimization. Recent methods leverage coverage criteria, adversarial testing, and execution traces to improve the discovery of unsafe behaviors Pei et al. (2017); Tian et al. (2018); Szegedy et al. (2013); Kurakin et al. (2018); Zhang et al. (2022); Lyu et al. (2025b;a). However, dynamic testing cannot provide formal safety guarantees in the absence of counterexamples. Consequently, integrating dynamic falsification with static verification offers a promising direction for improving both verification effectiveness and scalability.

6 Conclusion

This paper has presented *SynVAR*, a synergistic framework integrating dynamic falsification with static reachability analysis for NNCSs. By incorporating backward reachability preprocessing, PGD-guided adversarial testing, and BaB refinement verification, *SynVAR* improves counterexample search capability while reducing verification conservativeness. In particular, backward analysis focuses verification on critical regions, whereas adversarial-guided refinement tightens uncertain branches through targeted partitioning. Experimental results demonstrate the effectiveness and scalability in improving verification precision and determinacy.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide the extended problem formulation and preliminaries in Appendix A.1, followed by the complete *SynVAR* algorithm and soundness analysis in Appendix A.2. Detailed methodological descriptions, including backward reachability preprocessing, PGD-based adversarial falsification, and adversarially guided critical time-step and neuron selection, are provided in Appendix A.3. Detailed experimental settings, implementation configurations, additional case studies, runtime experiments, adversarial testing results, and complete benchmark specifications are reported in Appendix A.4. Finally, the limitations of the proposed framework and promising directions for future work are discussed in Appendix A.5. Implementation details, together with the code repository containing the source code, scripts, and configuration files required to reproduce the reported results, are available at `https://anonymous.4open.science/r/wxy-F37A/`.

REFERENCES

Matthias Althoff. An introduction to cora 2015 (tool presentation). In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, volume 2015, pp. 1–28.

Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. Juliareach: a toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 39–44, 2019.

Taha Entesari and Mahyar Fazlyab. Automated reachability analysis of neural network-controlled systems via adaptive polytopes. In *Learning for Dynamics and Control Conference*, pp. 407–419. PMLR, 2023.

Taha Entesari, Sina Sharifi, and Mahyar Fazlyab. Reachlipbnb: A branch-and-bound method for reachability analysis of neural autonomous systems using lipschitz bounds. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1003–1010. IEEE, 2023.

Michael Everett, Golnaz Habibi, and Jonathan P How. Robustness analysis of neural networks via efficient partitioning with applications in control systems. *IEEE Control Systems Letters*, 5(6): 2114–2119, 2020.

Michael Everett, Rudy Bunel, and Shayegan Omidshafiei. Drip: Domain refinement iteration with polytopes for backward reachability analysis of neural feedback loops. *IEEE Control Systems Letters*, 7:1622–1627, 2023.

Martin T Hagan, Howard B Demuth, and Orlando De Jesús. An introduction to the use of neural networks in control systems. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 12(11):959–985, 2002.

Akash Harapanahalli, Saber Jafarpour, and Samuel Coogan. Contraction-guided adaptive partitioning for reachability analysis of neural network controlled systems. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pp. 6044–6051. IEEE, 2023.

Haimin Hu, Mahyar Fazlyab, Manfred Morari, and George J Pappas. Reach-sdp: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming. In *2020 59th IEEE conference on decision and control (CDC)*, pp. 5929–5934. IEEE, 2020.

Kenneth J Hunt, Daniel Sbarbaro, R Żbikowski, and Peter J Gawthrop. Neural networks for control systems—a survey. *Automatica*, 28(6):1083–1112, 1992.

Saber Jafarpour, Akash Harapanahalli, and Samuel Coogan. Interval reachability of nonlinear dynamical systems with neural network controllers. In *Learning for Dynamics and Control Conference*, pp. 12–25. PMLR, 2023.

Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. Open-and closed-loop neural network verification using polynomial zonotopes. In *NASA Formal Methods Symposium*, pp. 16–36. Springer, 2023.

Suhas Kotha, Christopher Brix, J Zico Kolter, Krishnamurthy Dvijotham, and Huan Zhang. Provably bounding neural network preimages. *Advances in Neural Information Processing Systems*, 36: 80270–80290, 2023.

Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pp. 99–112. Chapman and Hall/CRC, 2018.

Tobias Ladner and Matthias Althoff. Exponent relaxation of polynomial zonotopes and its applications in formal neural network verification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 21304–21311, 2024.

Diego Manzanas Lopez, Sung Woo Choi, Hoang-Dung Tran, and Taylor T Johnson. Nnv 2.0: the neural network verification tool. In *International Conference on Computer Aided Verification*, pp. 397–412. Springer, 2023.

Diego Manzanas Lopez, Matthias Althoff, Luis Benet, Samuel Coogan, Marcelo Forets, Akash Harapanahalli, Taylor T. Johnson, Tobias Ladner, Christian Schilling, Huan Zhang, and Xiangru Zhong. Arch-comp25 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In Goran Frehse and Matthias Althoff (eds.), *Proceedings of 12th Int. Workshop on Applied Verification for Continuous and Hybrid Systems*, volume 108 of *EPiC Series in Computing*, pp. 71–121. EasyChair, 2025. doi: 10.29007/9vg6. URL /publications/paper/Gc39.

Deyun Lyu, Yi Li, Zhenya Zhang, Paolo Arcaini, Xiao-Yi Zhang, Fuyuki Ishikawa, and Jianjun Zhao. Fault localization of ai-enabled cyber–physical systems by exploiting temporal neuron activation. *Journal of Systems and Software*, 229:112475, 2025a.

Deyun Lyu, Zhenya Zhang, Paolo Arcaini, Xiao-Yi Zhang, Fuyuki Ishikawa, and Jianjun Zhao. Spectacle: Fault localisation of ai-enabled cps by exploiting sequences of dnn controller inferences. *ACM Transactions on Software Engineering and Methodology*, 34(4):1–35, 2025b.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Diego Manzanas Lopez, Matthias Althoff, Luis Benet, Clemens Blab, Marcelo Forets, Yuhao Jia, Taylor T Johnson, Manuel Kranzl, Tobias Ladner, Lukas Linauer, et al. Arch-comp24 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In *EPiC Series in Computing*, 2024.

Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*, pp. 1–18, 2017.

Nicholas Rober. Breach-lp: a framework for backward reachability analysis of neural feedback loops. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, 2023.

Nicholas Rober and Jonathan P How. Constraint-aware refinement for safety verification of neural feedback loops. *IEEE Control Systems Letters*, 2024.

Nicholas Rober, Michael Everett, and Jonathan P How. Backward reachability analysis for neural feedback loops. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 2897–2904. IEEE, 2022.

Nicholas Rober, Michael Everett, Songan Zhang, and Jonathan P How. A hybrid partitioning strategy for backward reachability of neural feedback loops. In *2023 American Control Conference (ACC)*, pp. 3523–3528. IEEE, 2023a.

Nicholas Rober, Sydney M Katz, Chelsea Sidrane, Esen Yel, Michael Everett, Mykel J Kochenderfer, and Jonathan P How. Backward reachability analysis of neural feedback loops: Techniques for linear and nonlinear systems. *IEEE Open Journal of Control Systems*, 2:108–124, 2023b.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*. ACM, 2018.

Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. Nnv: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, pp. 3–17. Springer, 2020.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 2002.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

Xian-Ming Zhang, Qing-Long Han, Xiaohua Ge, Derui Ding, Lei Ding, Dong Yue, and Chen Peng. Networked control systems: A survey of trends and techniques. *IEEE/CAA Journal of Automatica Sinica*, 7(1):1–17, 2019.

Zhenya Zhang, Deyun Lyu, Paolo Arcaini, Lei Ma, Ichiro Hasuo, and Jianjun Zhao. Falsifai: Falsification of ai-enabled hybrid control systems guided by time-aware coverage criteria. *IEEE Transactions on Software Engineering*, 49(4):1842–1859, 2022.

A Appendix

In the main text, we propose a synergistic verification framework *SynVAR* for neural network controlled systems, that integrates static reachability-based verification with projected gradient descent (PGD) guided dynamic falsification in a unified loop, combining the formal safety guarantees of static analysis with the efficient counterexample discovery capability of dynamic testing. In particular, *SynVAR* leverages backward reachability analysis and adversarial trajectories generated by PGD to guide targeted branch-and-bound refinement over critical time steps and neurons, thereby reducing conservativeness and improving verification efficiency.

In the appendix, we aim to address potential questions from readers through the following sections. These sections include, but are not limited to, further clarifying the applicability of our method *SynVAR*, presenting benchmark details and experimental settings, and providing additional experimental results to demonstrate the effectiveness of *SynVAR* for runtime application.

A.1 Extended Problem Formulation and Preliminaries

A.1.1 Neural Network Controlled Systems

Neural network controlled systems (NNCSs) with neural network controllers consist of a physical plant and a neural-network-based controller. The plant typically evolves according to continuous-time dynamics, while the controller operates in discrete time by periodically sampling the current system state and generating control inputs. The overall structure of the system is illustrated in Fig. 4.

For continuous-time modeling, the controlled plant dynamics in Fig. 4 can be written as:

$$\dot{x}(t) = f_c(x(t), u(t)), \quad x(t) \in \mathcal{X},\ u(t) \in \mathcal{U} \tag{4}$$

When the controller samples the system state with a sampling period $\tau > 0$ and holds the control input constant between adjacent sampling instants, the closed-loop evolution at sampling times can be represented by the following discrete-time dynamics:

$$x_{k+1} = F_\tau(x_k, u_k), \quad u_k = \sigma(x_k) \tag{5}$$

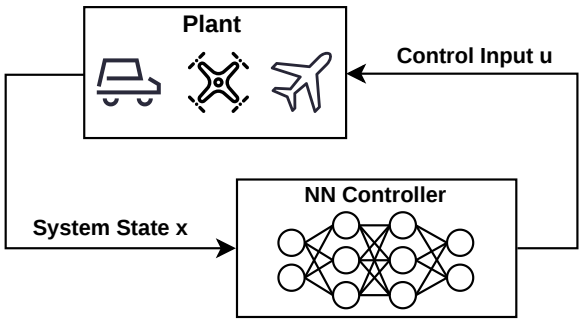


Figure 4. The structure of neural network controlled systems

where F_τ denotes the discrete transition induced by the continuous dynamics over the interval τ, and σ denotes the neural network controller. To emphasize the core problems studied in formal verification, this paper adopts the following simplified discrete-time formulation:

$$\begin{cases} x(t+1) = f(x(t), u(t)), & t \geq 0 \\ x(0) = x_0, & t = 0 \end{cases} \tag{6}$$

where u denotes the control input, the system state x serves as the input to the neural network controller, and f defines the discrete-time dynamics of the controlled plant.

Neural network controllers are typically implemented as feedforward fully connected networks with nonlinear activation functions. Let $n(\cdot)$ denote the input-output mapping of the neural network. An L-layer neural network can be denoted as:

$$n(x) = n_L \circ n_{L-1} \circ \cdots \circ n_1(x) \tag{7}$$

where

$$n_i(x) = \sigma(W_i x + b_i) \tag{8}$$

denotes the mapping of the i-th layer after applying the activation function σ to the affine transformation, with W_i and b_i representing the weight matrix and bias vector.

The execution process of the entire NNCS alternates among sampling, control, and system evolution. Given an initial state set and a sampling period τ, at the k-th sampling instant, the controller observes the current state x_k and generates the control input $u_k = \sigma(x_k)$. The plant then evolves under this control input to the next sampling instant, producing the subsequent state x_{k+1}. The objective of formal verification is to exhaustively analyze all possible initial states and closed-loop trajectories within a specified time horizon, in order to determine whether the system satisfies the required safety property specifications.

A.1.2 Safety Definition of NNCSs

The reachable set of NNCSs at time t characterizes all possible system states that can be reached from the initial states under the closed-loop dynamics, denoted by $\mathcal{R}_t$. In the discrete-time setting, given the initial state x_0, the reachable set can be interpreted as the set of states corresponding to time step t generated by the state transition sequence $\{x_0, x_1, \ldots, x_t\}$. The core problem of safety verification is to determine whether the reachable set of NNCSs intersects with the unsafe region $\mathcal{X}_{\text{unsafe}}$ under a given time condition (e.g., within a finite time horizon). If the two sets never intersect, then system is considered safe.

Formally, let $\mathcal{R}_i$ denote the reachable set over the time interval $[t_i, t_{i+1}]$. As shown in Fig. 5, if the NNCS is *"Safe"* within this interval, the following condition should hold:

$$\mathcal{R}_i \cap \mathcal{X}_{\text{unsafe}} = \emptyset. \tag{9}$$

If the NNCS is *"Unsafe"* within this time interval (i.e., some reachable states of the system enter the unsafe region), then the reachable set satisfies:

$$\mathcal{R}_i \subseteq \mathcal{X}_{\text{unsafe}} \tag{10}$$

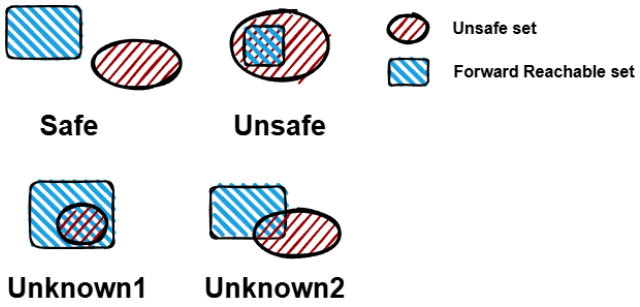


Figure 5. Verification Results of NNCSs

If the verification result is *"Unknown"* (e.g., due to the conservativeness introduced by over-approximation or limited numerical precision), the reachable set intersects with the unsafe region but is not entirely contained within it:

$$\mathcal{R}_i \cap \mathcal{X}_{\text{unsafe}} \neq \emptyset \quad \text{and} \quad \mathcal{R}_i \not\subseteq \mathcal{X}_{\text{unsafe}} \tag{11}$$

The above three cases correspond to the verification outcomes of safe, unsafe, and unknown, respectively. Among them, the *"Unknown"* result usually indicates that the current reachable set computation or verification method cannot provide sufficiently tight set bounds to draw a definitive conclusion. In such cases, further refinement of reachable set estimation and reduction of approximation errors are required to obtain more precise verification results.

In verification paradigms dominated by reachability analysis, it is common to distinguish between over-approximated and under-approximated reachable sets. Let $\widehat{\mathcal{R}}_t$ denote an over-approximation of the true reachable set $\mathcal{R}_t$, satisfying

$$\mathcal{R}_t \subseteq \widehat{\mathcal{R}}_t \tag{12}$$

If $\widehat{\mathcal{R}}_t \cap \mathcal{X}_{\text{unsafe}} = \emptyset$ holds for all time steps, then the system is guaranteed to be safe within the considered time horizon; this conclusion is sound. Conversely, if a concrete trajectory or an under-approximated reachable subset can be shown to enter $\mathcal{X}_{\text{unsafe}}$, then a counterexample (cex) demonstrating unsafety can be provided.

The *"Unknown"* verification outcome is typically caused by overly conservative over-approximations that lead to spurious intersections. To address this issue, tighter abstract domains, finer state-space partitioning strategies, or additional constraint information are often introduced to further refine the safety verification process.

A.1.3 Goal-Reach Specification

In the main text, we introduced the *reach-avoid* safety property, which requires that closed-loop trajectories avoid entering the unsafe set $\mathcal{X}_{\text{unsafe}}$ over a given time horizon. In addition to this class of specifications, we also consider a class of *goal-reach* tasks.

The verification of the goal-reach property is conducted based on reachable set analysis. Specifically, at a given time step k, if the reachable sets of all branches can be proven to be entirely contained within the target set (i.e., $\mathcal{R}_k \subseteq \mathcal{X}_{\text{goal}}$), then the property is certified as *safe*. If the current over-approximation can only establish that a subset of branches lie within $\mathcal{X}_{\text{goal}}$, while there still exist branches that cannot be proven to be fully contained in $\mathcal{X}_{\text{goal}}$, the result is returned as *"Unknown"*. In this case, branch-and-bound (BaB) refinement and boundary tightening are further performed until either a definitive conclusion is reached or the computational budget is exhausted.

During PGD-guided sample generation, the attack objective for goal-reach task is opposite to that of reach-avoid. Specifically, the loss is designed as a measure of distance away from target set, so as to drive the search toward adversarial initial states whose trajectories fail to reach the goal.

Concretely, we adopt the point-to-set distance $d(\cdot, \mathcal{X}_{\text{goal}})$ and define the loss function

$$\mathcal{L}_{\text{goal}}(x_0) = \min_{k \in \{0,\dots,N\}} d\big(F_k(x_0), \mathcal{X}_{\text{goal}}\big), \tag{13}$$

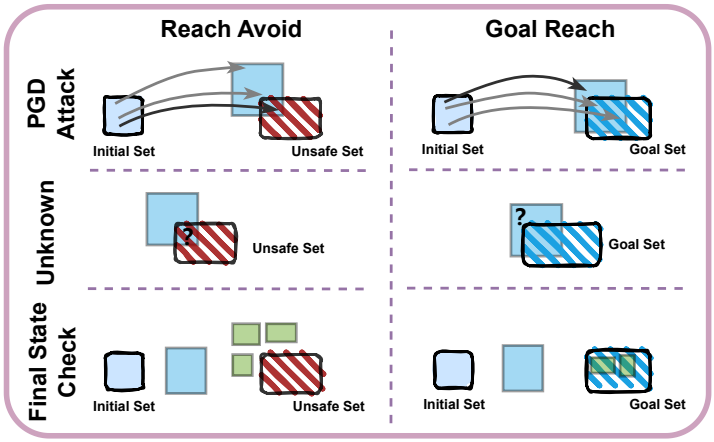


Figure 6. Difference between reach-avoid and goal-reach

where $\mathcal{L}_{\text{goal}}(x_0) = 0$ indicates that the trajectory reaches the target set at some time step. The PGD procedure maximizes $\mathcal{L}_{\text{goal}}$ to push the trajectory away from $\mathcal{X}_{\text{goal}}$ as a whole.

Accordingly, safe samples are those whose trajectories satisfy the reach condition, i.e., enter $\mathcal{X}_{\text{goal}}$. In contrast, adversarial samples are those that lie away from the boundary of the target set. Although such trajectories may still reach the goal, they are sensitive to perturbations and exhibit a high risk of deviating away from $\mathcal{X}_{\text{goal}}$. These samples are further utilized to guide the selection of critical time steps and neurons, as well as subsequent BaB refinement.

A.1.4 Dynamic Falsification via Adversarial Testing

Dynamic falsification seeks to discover counterexamples by searching for system executions that violate given safety specifications. For NNCSs, this problem can be formulated as an optimization problem over initial states or input parameters, where a violation objective is maximized to identify adversarially sensitive trajectories.

Specifically, given an NN controller $f_\theta : \mathbb{R}^n \to \mathbb{R}^k$ and an input $x \in \mathbb{R}^n$, adversarial testing seeks a perturbed input $x' = x + \delta$ that maximizes violation of a given specification. This can be formulated as

$$\begin{aligned} \max_{\delta} \quad & L(f_\theta(x+\delta)) \\ \text{s.t.} \quad & \|\delta\|_p \le \epsilon, \end{aligned}$$

where $L(\cdot)$ is a differentiable objective that quantifies violation of safety or performance constraints, and ϵ bounds the perturbation magnitude. Here, we employ projected gradient descent (PGD Madry et al. (2017)) optimization method to search the initial state space by exploiting trajectory-level gradients. However, dynamic falsification only provides concrete counterexamples when violations are found and cannot establish formal safety guarantees in their absence.

A.2 Soundness Proof of *SynVAR*

A.2.1 Overview of *SynVAR* Framework

In the context, we propose a synergistic framework named *SynVAR*, as illustrated in Fig. 3. The framework integrates dynamic falsification with static verification in a unified iterative loop, and can be organized into three components: backward reachability preprocessing, PGD adversarial falsification, and adversarial-guided forward reachability verification.

Concretely, the process begins with backward reachability analysis to derive time-indexed safety constraints and refine the initial state set, thereby reducing the search space for subsequent analysis. Building upon this, adversarial attacks are performed to efficiently explore potential failure behav-

Algorithm 1 Synergistic **V**erification of NNCSs via **A**dversarially Guided **R**efinement (***SynVAR***)

Input: Neural network controller nn, system dynamics f, initial state set $\mathcal{X}_0$, unsafe set $\mathcal{X}_{\text{unsafe}}$, time horizon T, branch-and-bound budget B
Output: Result $\mathcal{S} \in \{\textit{Safe}\checkmark, \textit{Unsafe}\times, \textit{Unknown}\}$
1: $\{\mathcal{C}_t\}_{t=0}^{T} \leftarrow \text{BackwardReach}(\mathcal{X}_{\text{unsafe}})$
2: $\widetilde{\mathcal{X}}_0 \leftarrow \mathcal{X}_0 \cap \mathcal{C}_T$
3: $(\mathcal{P}_{\text{safe}}, \mathcal{P}_{\text{pot}}, \mathcal{P}_{\text{unsafe}}) \leftarrow \text{PGDAttack}(\widetilde{\mathcal{X}}_0, nn, f, \mathcal{X}_{\text{unsafe}}, T, B)$
4: **if** $\mathcal{P}_{\text{unsafe}} \neq \emptyset$ **then**
5: **return** *Unsafe*✗
6: **end if**
7: $\mathcal{S} \leftarrow \textit{Unknown}$
8: **while** $\mathcal{S} = \textit{Unknown}$ **do**
9: $\mathcal{S} \leftarrow \text{ForwardReachAndCheck}(nn, f, \mathcal{X}_0, \mathcal{X}_{\text{unsafe}}, T)$
10: **if** $\mathcal{S} \neq \textit{Unknown}$ **then**
11: **return** $\mathcal{S}$
12: **end if**
13: $\mathcal{S} \leftarrow \text{BaB_Refinement}(\mathcal{P}_{\text{safe}}, \mathcal{P}_{\text{pot}})$
14: **end while**
15: **return** $\mathcal{S}$

iors by minimizing the distance between system trajectories and the unsafe region. The resulting trajectories, which include both safe and potential unsafe (adversarial) cases, are then exploited to extract guidance signals. In particular, discrepancies in neuron activations and the accumulation of overapproximation errors across time steps are analyzed to identify critical time steps and neurons, indicating where refinement is most needed. Finally, forward reachability analysis is performed to verify the safety property. When it yields an *"Unknown"* result, the framework selectively performs branch-and-bound refinement at the identified critical regions by partitioning key neuron domains.

The overall framework operates iteratively and leverages targeted adversarially guided refinement to mitigate overapproximation errors and improve verification precision, consisting of the following three stages:

- **Backward preprocessing.** Time-indexed safety constraints are derived to refine the initial state set, thereby reducing the search space for subsequent analysis.
- **PGD adversarial falsification.** Adversarial exploration is employed to search for potential unsafe behaviors, while the resulting trajectories are further exploited to identify critical time steps and neurons based on activation discrepancies.
- **Reachability analysis verification.** Forward reachable set propagation is performed to verify safety properties. When the verification result remains inconclusive (*"Unknown"*), branch-and-bound (BaB) refinement is selectively applied to critical neurons and time steps, thereby reducing overapproximation errors and improving verification precision.

Recall Algorithm 1, which presents the complete verification procedure of *SynVAR*. Lines 1-2 describe the backward reachability analysis, which computes a sequence of backward reachable sets $\{\mathcal{C}_t\}_{t=0}^{T}$. By intersecting the final reachable set with $\mathcal{X}_0$, a refined initial state space $\widetilde{\mathcal{X}}_0$ is obtained. Line 3 specifies that safe $\mathcal{P}_{\text{safe}}$, unsafe $\mathcal{P}_{\text{unsafe}}$, and adversarial samples $\mathcal{P}_{\text{pot}}$ are generated via the PGD-based attack. If the PGD attack has already identified an unsafe sample, the algorithm immediately returns *"Unsafe"*, and the verification process terminates. Otherwise, the system status is initialized as *"Unknown"*. Lines 8-14 describe the main iterative verification loop. As long as no deterministic verification result is obtained, the following steps are repeated. Forward reachability analysis combined with property checking determines the current system status in Line 9. If the result is either *"Safe"* or *"Unsafe"*, the verification process terminates. Otherwise, the status remains *"Unknown"*, and a refinement step is performed via the branch-and-bound method in Line 13.

Theorem 1 (Soundness of *SynVAR*) *Assume that (i) the forward reachability procedure computes sound overapproximations of the exact reachable sets, (ii) the backward reachability procedure computes sound overapproximations of predecessor sets, and (iii) each branch-and-bound (BaB) refine-*

ment partitions its parent domain into disjoint subdomains whose union equals the parent domain. Then SynVAR is sound, i.e., it returns SAFE *only if the closed-loop system satisfies the safety specification, and returns* UNSAFE *only if a concrete counterexample exists.*

Proof 1 *Let $R^*(X)$ denote the exact reachable set of an initial set X, and let $\widehat{R}(X)$ denote the reachable-set overapproximation maintained by SynVAR.*

Backward preprocessing reconstructs the verification domain as

$$X_e = X_0 \cap C_T,$$

where C_T is a sound predecessor overapproximation of the unsafe set. Hence every initial state that may reach X_{unsafe} is preserved, i.e.,

$$R^*(X_0) \cap X_{\text{unsafe}} \neq \varnothing \implies R^*(X_e) \cap X_{\text{unsafe}} \neq \varnothing.$$

PGD-based adversarial search is used only to search for concrete counterexamples and to prioritize BaB refinement. If PGD returns an unsafe trajectory, the trajectory is generated by the original closed-loop system and therefore constitutes a genuine counterexample. Otherwise, the verification domain remains unchanged.

The subsequent forward reachability analysis computes a sound overapproximation satisfying

$$R^*(X) \subseteq \widehat{R}(X).$$

Therefore,

$$\widehat{R}(X) \cap X_{\text{unsafe}} = \varnothing \implies R^*(X) \cap X_{\text{unsafe}} = \varnothing,$$

which justifies every SAFE *verdict.*

For every UNKNOWN *node, BaB recursively partitions the verification domain into*

$$X = \bigcup_{i=1}^{m} X_i, \qquad X_i \cap X_j = \varnothing,\ i \neq j.$$

Hence,

$$R^*(X) = \bigcup_{i=1}^{m} R^*(X_i),$$

and no reachable behavior is discarded during refinement. Since each child node is verified using the same sound reachability procedure, the aggregation rule returns SAFE *only if every child is safe, and returns* UNSAFE *whenever a genuine counterexample is found.*

Therefore, every operation in SynVAR preserves the soundness of the verification process. By composition, the entire algorithm is sound: every SAFE *verdict certifies the absence of unsafe reachable states, while every* UNSAFE *verdict corresponds to a concrete counterexample.*

A.3 ADDITIONAL METHOD DETAILS

A.3.1 BACKWARD REACHABILITY PREPROCESSING AND INITIAL SET RECONSTRUCTION

A major challenge in static verification arises from the accumulation of overapproximation errors across time steps. To filter out initial states irrelevant to unsafe behaviors before verification, the proposed framework first conducts backward reachability analysis. Starting from $\mathcal{X}_{\text{unsafe}}$, the method recursively computes overapproximated predecessor sets that can potentially lead to the unsafe region at each time step, and represents them as backward safety constraints $\{\mathcal{C}_t\}_{t=0}^{T}$.

Backward reachability preprocessing plays a critical role in guiding the subsequent verification modules in several aspects:

[Initial State Space Reconstruction]. The set $\mathcal{C}_T$ characterizes the subset of initial states that may drive the closed-loop system into unsafe regions during execution. Based on this observation, we refine the initial set as

$$\widetilde{\mathcal{X}}_0 = \mathcal{X}_0 \cap \mathcal{C}_T,$$

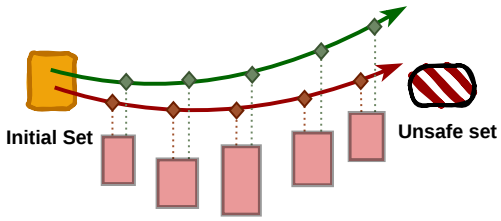


Figure 7. Guidance effect of backward reachability preprocessing on PGD

thereby restricting subsequent analysis to safety-critical regions relevant to unsafe behaviors.

By excluding initial states that can never reach the unsafe region, the refined set $\widetilde{\mathcal{X}}_0$ significantly reduces the size of reachable sets that need to be propagated.

Remark 1 *The constraint sets $\{\mathcal{C}_t\}$ provide a temporally consistent characterization of safety-critical regions $\widetilde{\mathcal{X}}_0$ and can be systematically reused across different components of the framework. They enable structured guidance for PGD-based falsification by supplying time-indexed constraints, which induce dense trajectory-level optimization signals rather than relying solely on terminal objectives. Furthermore, they can facilitate branch-and-bound verification by serving as auxiliary filtering criteria: subregions that do not intersect with the corresponding $\{\mathcal{C}_t\}$ can be pruned early, thereby reducing redundant partitioning and improving verification efficiency.*

[Guidance for PGD-Based Counterexample Search]. The constraint sets $\mathcal{C}_t$ can be regarded as progressively defined unsafe postconditions, providing a temporally structured guidance mechanism for trajectory-level adversarial search. Unlike optimization approaches that only rely on terminal objectives, the PGD process is guided at every time step toward regions satisfying the constraints $\mathcal{C}_t$, thereby improving both convergence efficiency and attack success rate. As illustrated in Fig. 7, the red diamonds denote adversarial samples at different time steps, while the green diamonds represent safe samples. By computing the distances between these samples and the backward reachable sets, the PGD trajectory is iteratively guided toward or away from unsafe regions, enabling directional search and update behaviors.

[Auxiliary Refinement for BaB Verification]. Within the BaB framework, the same constraint sets $\mathcal{C}_t$ can be reused for intermediate branch safety screening. For subregions that remain classified as *"Unknown"* after multiple partitioning steps, if they do not intersect with the corresponding backward reachable sets, they can be reliably verified as safe. This constraint structure enables early pruning of infeasible branches, avoids redundant partitioning operations, and further improves overall verification efficiency.

A.3.2 Detailed Description of PGD-based Adversarial Attack for NNCSs

In this section, we introduce a principle of gradient-based optimization for adversarial falsification on NNCSs to efficiently explore potential unsafe behaviors in closed-loop systems. Since safety violations depend on the long-horizon propagation of the initial state through both the system dynamics and neural network controller, sampling-based black-box approaches often suffer from inefficient exploration. Importantly, the differentiability of the closed-loop dynamics with respect to the initial state enables gradient-based analysis of how perturbations propagate through system trajectories over time.

Motivated by this observation, we reformulate falsification as an optimization problem over the initial state space. Specifically, a trajectory-level violation objective is constructed to characterize the proximity between system trajectories and the unsafe region. PGD is then to iteratively update the initial state samples toward adversarially sensitive regions, thereby improving the efficiency of unsafe trajectory exploration.

To characterize trajectory evolution, we define the k-step state mapping $F_k : \mathbb{R}^n \to \mathbb{R}^n$ such that $x_k = F_k(x_0)$. To obtain adversarial and safe samples, a signed distance function is introduced to

measure the proximity between a state and the unsafe region:

$$d(x, \mathcal{X}_{\text{unsafe}}) = \begin{cases} -\inf_{y \in \mathcal{X}_{\text{unsafe}}} \|x - y\|_2 & x \notin \mathcal{X}_{\text{unsafe}}, \\ \inf_{y \in \mathbb{R}^n \setminus \mathcal{X}_{\text{unsafe}}} \|x - y\|_2 & x \in \mathcal{X}_{\text{unsafe}}. \end{cases} \tag{14}$$

Based on (14), the trajectory risk over horizon N can be naturally quantified as:

$$\mathcal{R}(x_0) = \max_{k \in \{0,\dots,N\}} d\left(F_k(x_0), \mathcal{X}_{\text{unsafe}}\right). \tag{15}$$

Maximizing (15) drives the trajectory toward or into the unsafe region, thus facilitating counterexamples discovery.

In practice, the unsafe region is typically characterized by a set of inequality constraints:

$$\mathcal{X}_{\text{unsafe}} = \{x \in \mathbb{R}^n \mid g_i(x) \le 0,\ i = 1, \dots, s\}.$$

However, directly obtaining $\mathcal{R}(x_0)$ in (15) is often computationally expensive. To enable efficient optimization, we instead introduce the following constraint violation function:

$$V(x) = -\max_{i \in \{1,\dots,s\}} g_i(x),$$

which serves as a differentiable surrogate objective for measuring proximity to the unsafe region. Accordingly, the trajectory-level loss is approximated by

$$J_V(x_0) = \max_{k \in \{0,\dots,N\}} V(F_k(x_0)). \tag{16}$$

The differentiability of objective (16) enables the use of gradient-based optimization for adversarial exploration. Here, PGD is employed as an efficient first-order optimization method to search for adversarially sensitive initial states under bounded constraints. Specifically, the initial state is iteratively updated to maximize the trajectory-level violation objective while enforcing the feasibility constraint $x_0 \in \mathcal{X}_0$, where $\mathcal{X}_0$ is assumed to be a hyper-rectangle:

$$\mathcal{X}_0 = \{x \in \mathbb{R}^n \mid l_i \le x[i] \le u_i,\ i = 1, \dots, n\}.$$

At iteration t, the PGD update is performed as

$$\begin{cases} g^{(t)} = \nabla_{x_0} J_V(x_0)\big|_{x_0 = x_0^{(t)}}, \\ x_0^{(t+\frac{1}{2})} = x_0^{(t)} + \alpha \cdot \operatorname{sign}(g^{(t)}), \\ x_0^{(t+1)} = \operatorname{Clip}_{\mathcal{X}_0}\left(x_0^{(t+\frac{1}{2})}\right), \end{cases}$$

where the gradient is computed by backpropagation through time (BPTT Werbos (2002)). The projection operator $\operatorname{Clip}_{\mathcal{X}_0}(\cdot)$ projects the updated state back to the feasible initial set:

$$\operatorname{Clip}_{\mathcal{X}_0}(x)[i] = \min(\max(x[i], l_i), u_i), \quad i = 1, \dots, n.$$

Notably, unlike conventional PGD designed for single-step classification models, the optimization variable in this setting is the initial state x_0, and the loss function (16) is defined over the entire closed-loop trajectory within the time horizon N. The key idea is to iteratively optimize x_0 within the initial set $\mathcal{X}_0$ by maximizing a trajectory-level risk metric, thereby searching for initial conditions that may drive the system into the unsafe region. Consequently, our adversarial PGD process can be divided into the following four steps:

- **Step1: objective for loss function:** The attack maximizes the trajectory-based loss $J_V(x_0)$, which encourages trajectories to approach or enter the unsafe set $\mathcal{X}_{\text{unsafe}}$.
- **Step2: forward simulation and safety checking**: At each iteration, PGD performs forward simulation to generate a trajectory

  $$\mathcal{P}^{(t)} = \{x_k^{(t)}\}_{k=0}^{N}.$$

  It then checks whether any state along the trajectory violates the safety condition. If a genuine trajectory that violates the system's security state is detected, the verification process is immediately terminated and an *"Unsafe"* verification result is returned.

- **Step3: gradient computation and PGD update:** If no violation is found, the gradient of the loss function with respect to the initial state is computed via BPTT and fomulated as:

$$g^{(t)} = \nabla_{x_0} J_V(x_0)\big|_{x_0=x_0^{(t)}}.$$

  The initial state is then updated using a PGD step, followed by projection onto the feasible set $\mathcal{X}_0$ to ensure constraint satisfaction.
- **Step4: multiple restarts and trajectory selection:** To mitigate local optima, multiple random restarts are employed. If no counterexample is found, the trajectory with the highest loss value is selected as the potentially unsafe trajectory $\mathcal{P}_{\text{pot}}$, while the trajectory farthest from the unsafe region is selected as the representative safe trajectory $\mathcal{P}_{\text{safe}}$, both of which are used to guide subsequent verification and refinement.

Algorithm 2 PGD-based adversarial attack for NNCSs

Input: System dynamics f, neural network controller nn, initial set $\mathcal{X}_0$, unsafe set $\mathcal{X}_{\text{unsafe}}$, time horizon N, number of random restarts R, number of PGD iterations K, step size α
Output: $\mathcal{P}_{\text{safe}},\ \mathcal{P}_{\text{pot}},\ \mathcal{P}_{\text{unsafe}}$
1: Initialize $\mathcal{P}_{\text{safe}},\ \mathcal{P}_{\text{pot}},\ \mathcal{P}_{\text{unsafe}}$
2: **for** $r = 1$ to R **do**
3: $\quad x_0 \leftarrow \text{Sample}(\mathcal{X}_0)$
4: $\quad$ **for** $k = 1$ to K **do**
5: $\quad\quad \mathcal{P} \leftarrow \text{ForwardSimulate}(f, nn, x_0, N)$
6: $\quad\quad$ **if** $\text{IsUnsafe}(\mathcal{P}, \mathcal{X}_{\text{unsafe}})$ **then**
7: $\quad\quad\quad \mathcal{P}_{\text{unsafe}} \leftarrow \mathcal{P}_{\text{unsafe}} \cup \mathcal{P}$
8: $\quad\quad\quad$ **return** $\mathcal{P}_{\text{unsafe}}$
9: $\quad\quad$ **end if**
10: $\quad\quad (\mathcal{P}_{\text{safe}}, \mathcal{P}_{\text{pot}}) \leftarrow \text{UpdateBestTraj}(\mathcal{P}, \mathcal{P}_{\text{safe}}, \mathcal{P}_{\text{pot}})$
11: $\quad\quad x_0 \leftarrow \text{PGDStep}(x_0, f, nn, \mathcal{X}_{\text{unsafe}}, \mathcal{X}_0, N, \alpha)$
12: $\quad$ **end for**
13: **end for**
14: **return** $(\mathcal{P}_{\text{safe}}, \mathcal{P}_{\text{pot}}, \mathcal{P}_{\text{unsafe}})$

Recall Algorithm 2, which presents the PGD-based adversarial attack procedure for NNCSs. During the R restart iterations, the algorithm first samples an initial state (Line 3). Then, for each of the K PGD iterations, forward simulation is performed to obtain a trajectory $\mathcal{P}$. If the trajectory enters the unsafe region, an unsafe trajectory $\mathcal{P}_{\text{unsafe}}$ is identified (Lines 6-9). Otherwise, the trajectory is used to update the best safe $\mathcal{P}_{\text{safe}}$ or adversarial trajectories $\mathcal{P}_{\text{pot}}$ (Line 10). Then, a PGD update is performed (Line 11). Upon termination, the algorithm returns three categories of trajectories.

Remark 2 *In this paper, we consider the safety verification problem, where potentially unsafe states can be viewed as adversarial factors affecting reachability analysis and are therefore referred to as adversarial samples. Notably, PGD does not guarantee the discovery of concrete counterexamples. Instead, it generates informative trajectories, including concrete unsafe trajectories and adversarial trajectories near the unsafe boundary. Such trajectories provide valuable guidance for subsequent verification procedures, for example by tightening reachable set overapproximations and assisting BaB refinement, thereby improving the overall effectiveness.*

A.3.3 Analysis of Critical Time-Step Selection for BaB

When selecting a method for identifying key time steps, we conducted a series of explorations; initially, we planned to base our decision on the divergence in neural activity at each time step. After obtaining the safe trajectory set $\mathcal{S}_{\text{safe}}$ and the potentially unsafe trajectory set $\mathcal{S}_{\text{pot}}$, we compute the activation divergence score $D(t)$ (defined later) for each time step and rank all time steps in descending order according to their divergence strengths. Intuitively, a larger $D(t)$ indicates that the controller exhibits more significant internal activation discrepancies between safe and potentially unsafe trajectories at time step t. Such discrepancies are more likely to contribute to *"Unknown"* verification outcomes caused by overapproximation errors. Therefore, the selection strategy for critical time steps can be categorized according to the distribution pattern of $D(t)$ values.

Peak-Dominant Pattern (Top-1 Significantly Larger): If the largest divergence score satisfies

$$D(t_{(1)}) \gg D(t_{(2)}),$$

for example,

$$D(t_{(1)})/D(t_{(2)}) \geq \gamma,$$

then the critical time step can be directly selected as $t^\star = t_{(1)}$. In this case, refinement efforts are concentrated on the single most influential time step.

Multi-Peak Pattern (Multiple Competitive Candidates): If several time steps exhibit similarly large divergence scores, e.g.,

$$\left|\left\{t \mid D(t) \geq \beta D(t_{(1)})\right\}\right| > 1,$$

then the Top-K_T candidates are selected to form the candidate set $\mathcal{T}^\star$. During the branch-and-bound process, these candidates are refined sequentially according to their divergence rankings. If ties remain, later time steps are prioritized to better address the accumulation of propagation errors over time.

Flat Pattern (Weak Overall Distinction): If the divergence scores are relatively uniform across time steps, indicating weak discriminative signals between safe and potentially unsafe trajectories, the Top-K_T time steps are selected as candidate refinement positions. In this setting, a "later-time-step priority" strategy can be adopted to improve coverage of regions where accumulated approximation errors become more significant.

Late-Dominant Pattern (Divergence Accumulates Over Time): If $D(t)$ exhibits a clear increasing trend along the temporal horizon, later time steps with larger divergence scores are preferentially selected for refinement. This strategy directly targets the dominant sources of *"Unknown"* results caused by multi-step error accumulation in closed-loop propagation.

The proposed method performs safety-guided branch variable selection by defining divergence scores based on activation-rate differences between safe and potentially unsafe trajectories. These scores are further used to rank critical time steps and select Top-K critical neurons for refinement. During the BaB process, ReLU sign constraints $z > 0$ and $z \leq 0$ are introduced for branching, while constraint-injected *CROWN* propagation together with infeasibility pruning is employed to improve both the tightness of reachable-set bounds and the overall verification efficiency.

Remark 3 *In this paper, we consider the safety verification problem, where potentially unsafe states can be viewed as adversarial factors affecting reachability analysis and are therefore referred to as adversarial samples. Notably, PGD does not guarantee the discovery of concrete counterexamples. Instead, it generates informative trajectories, including concrete unsafe trajectories and adversarial trajectories near the unsafe boundary. Such trajectories provide valuable guidance for subsequent verification procedures, for example by tightening reachable set overapproximations and assisting BaB refinement, thereby improving the overall effectiveness.*

A.4 Experimental Details

A.4.1 Experimental setup

We provide the YAML configuration files for running *SynVAR* on each benchmark in the `example_configs` folder of our code repository. These configuration files contain all required parameters, including initial region, unsafe region, BaB budget limit, PGD attack initialization settings, and other necessary configurations.

The implementation of *SynVAR* is mainly located in the `run_progressive_bab_with_config` file. Specifically, the PGD-based adversarial attack module for neural network controlled systems is implemented in the `pgd_closed_loop_attack` file, while the BaB verification module is implemented in the `bab_components` file. These modules correspond respectively to the three subsections of the Method section 3 in the main paper.

A.4.2 Case study of 6D quadrotor Rober et al. (2023b)

Here, we further consider a 6D quadrotor benchmark to evaluate the performance of *SynVAR* on high-dimensional cases.

Example 2 (6D quadrotor) *The system is modeled by the following linearized dynamics:*

$$x_{t+1} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} u_t.$$

The initial state set is defined as $[-4.9, -4.75] \times [0.1, 0.25] \times [2.25, 2.5] \times [0.95, 0.99] \times [-0.01, 0.01] \times [-0.01, 0.01]$*, and the unsafe region is given by* $[-1, 1] \times [-1, 1] \times [1.5, 3.5] \times [-1, 1] \times [-1, 1] \times [-1, 1]$*. The objective is to verify that the system remains safe over a time horizon of 5 steps.*

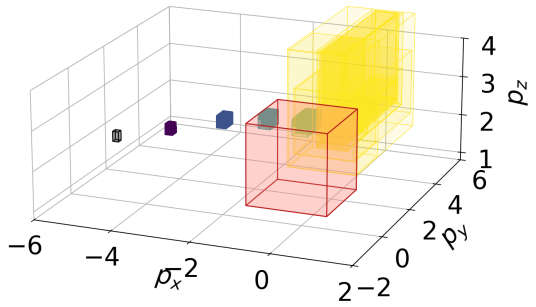


Figure 8. Verification results for Quadrotor visualized via 3D state-space projection. The gray rectangular region represents the initial state set of the system, while the red rectangular region denotes the unsafe set. The remaining rectangular regions correspond to the reachable state space partitions obtained via the branch-and-bound procedure over time, whose color gradually transitions from dark to light, indicating the evolution from the initial time step to the final time step.

As illustrated in Fig. 8, the reachable sets at different time steps evolve around the unsafe region while consistently remaining disjoint from it, thereby demonstrating the safety of the system. Moreover, all partitioned subregions generated by the BaB process are formally verified to have no intersection with the unsafe set. These results indicate that the branch-and-bound strategy effectively improves verification precision, while also showing that *SynVAR* remains capable of verifying safety properties in high-dimensional settings.

A.4.3 Effectiveness of Adversarially Guided Refinement

Fig. 9 illustrates the impact of neuron-wise BaB partitioning on NN output bounds for Example 1 and higher-dimensional 6D Quadrotor Rober et al. (2023b). In particular, we quantify the effectiveness of the refinement strategy by comparing the output-layer neuron bounds before and after neuron splitting. The vertical axis in the figure represents the boundary partitioning of the n-th neuron in the L_m-th layer of the control system at time step t_k.

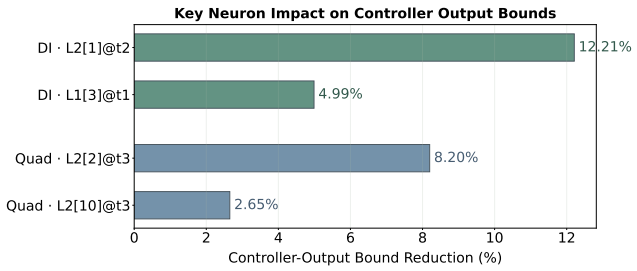


Figure 9. Comparison of NN output bounds before and after neuron splitting for the Double Integrator and 6D Quadrotor

The results show that adversarially guided partitioning can effectively tighten NN output bounds by identifying critical neurons that contribute most significantly to overapproximation reduction. Specifically, at time step t_2, the first neuron in the second layer achieves the largest improvement in bound precision after partitioning, reducing the bound range by 12.21%. These results demonstrate that the proposed refinement strategy can effectively reduce conservativeness in reachability analysis and improve verification precision.

A.4.4 Longer-Horizon Verification and Real-Time Safety Assurance

To further evaluate *SynVAR* beyond the verification horizons considered in the main experiments, we investigate its applicability to longer-horizon verification and real-time safety assurance. In particular, we consider a 30-step verification task and further examine the computational efficiency of both verification and falsification under short prediction horizons.

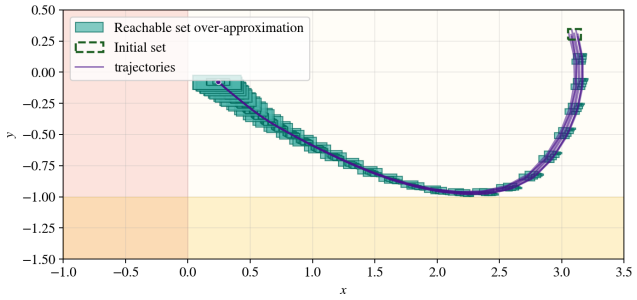


Figure 10. Verification process for the Double Integrator benchmark over a 30-step horizon.

We first consider the Double Integrator benchmark Rober & How (2024) over a horizon of 30 time steps. We set $dt = 0.2$, with the initial state set $[3.05, 3.15] \times [0.2625, 0.35]$, and impose the safety constraint $C = (x[0] \geq 0$ and $x[1] \geq -1)$. As shown in Fig. 10, the purple curves represent system trajectories, while the blue rectangles denote the reachable sets computed by *SynVAR* at successive time steps. The results illustrate *SynVAR* can propagate reachable sets over an extended prediction horizon and assess whether future system behaviors satisfy specified safety constraint. Such multi-step prediction provides a basis for applying reachability verification to runtime safety assessment.

We further evaluate the **computational efficiency** of *SynVAR* for runtime verification. On the Ground Robot benchmark, verification over horizons of 1, 2, and 3 steps requires only $0.09\,\mathrm{s}$, $0.11\,\mathrm{s}$, and $0.13\,\mathrm{s}$, respectively. These results indicate that short-horizon reachability analysis can be performed efficiently during system execution to assess future behaviors at control update steps. In addition, the PGD-based falsification procedure identifies an unsafe behavior after 6 time steps within $0.16\,\mathrm{s}$, demonstrating its ability to rapidly search for potential safety violations.

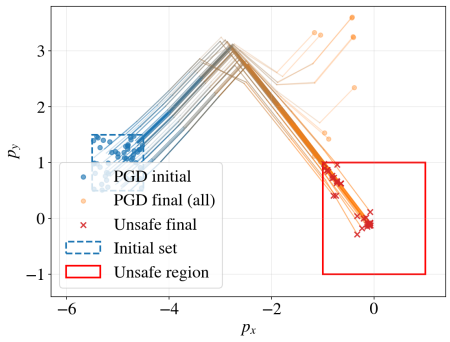


Figure 11. PGD-based falsification results for the Ground Robot buggy benchmark.

Fig. 11 illustrates the falsification process for the Ground Robot buggy benchmark. The blue dots denote the initial states sampled by the PGD-based attack, while the red dots represent the corre-

sponding states reached after the attack. The red rectangular region denotes the unsafe set. The generated trajectories reaching this region provide concrete unsafe behaviors, showing that the dynamic falsification component can efficiently detect safety violations when they exist.

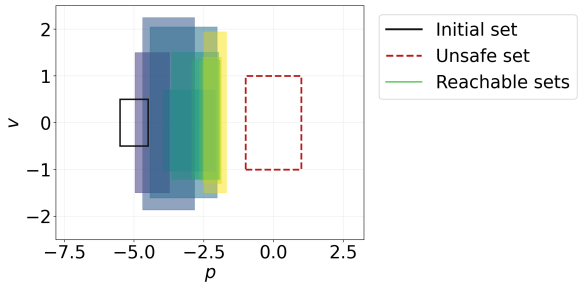
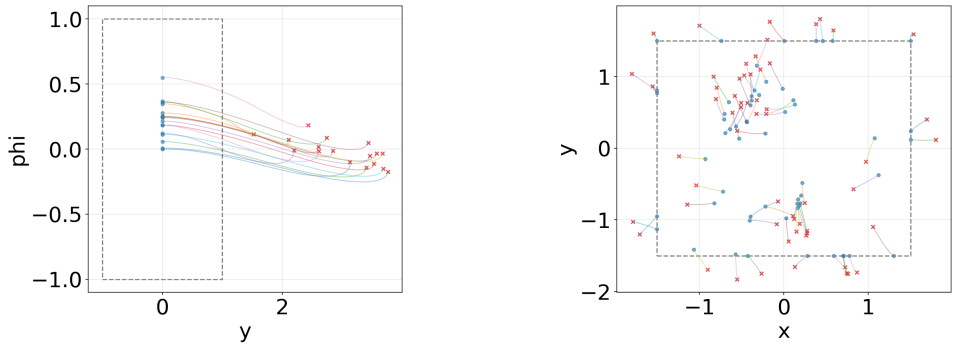


Figure 12. Safety verification process for the Ground Robot benchmark.

For the safe Ground Robot benchmark, Fig. 12 further illustrates how backward reachability preprocessing reduces unnecessary state-space exploration. The backward reachable set only partially intersects the original initial region, while the non-intersecting portion is excluded from subsequent analysis because it cannot reach the unsafe set within the specified horizon. *SynVAR* therefore focuses dynamic and static analysis on the reconstructed safety-critical initial region and applies adversarially guided BaB refinement when necessary. This targeted analysis reduces unnecessary exploration while retaining the soundness of the final verification result.

A.4.5 Performance of PGD-based Adversarial Testing Results

These results demonstrate that the PGD-based attack module in *SynVAR* can effectively guide the search process and generate a substantial number of adversarial samples that lead to unsafe system behaviors.



(a) Airplane (b) Double Pendulum

Figure 13. PGD-based adversarial testing results.

Fig. 13a and Fig. 13b illustrate the falsification process for the double inverted pendulum system and the airplane control system. In the figure, blue dots represent the initial states sampled by the PGD-based attack method, while red dots denote the resulting system states after the attack. The gray rectangular region corresponds to the safe state space, and any state falling outside this region is considered unsafe.

For the airplane control system benchmark, due to the high dimensionality of the initial state space, we project the states onto the ϕ-y plane for visualization. Since several dimensions of the initial input are fixed to zero, the sampled points exhibit a vertically aligned distribution. As observed from the figure, a large number of final states lie outside the safe region. For the double inverted pendulum system, the initial states that lead to unsafe behaviors are primarily located near the boundary of the safe region, which is consistent with intuitive expectations.

These results demonstrate that the PGD-based attack module in *SynVAR* can effectively guide the search process and generate a substantial number of adversarial samples that lead to unsafe system behaviors.

A.4.6 BENCHMARKS

This section provides an overview of the benchmarks mentioned in the main text that were not used as case studies.

[Double Pendulum System Lopez et al. (2025)]. The double pendulum system is a representative control benchmark with strong nonlinearity and coupling characteristics, and is widely used to evaluate the safety and stability of control systems under high-dimensional continuous dynamics. The system consists of two pendulum rods of equal length connected by joints, where each rod carries a point mass m at its end. The system operates around the upright equilibrium point under gravitational effects.

Let θ_1 and θ_2 denote the angular deviations of the two pendulums from the upright vertical direction, and let $\dot{\theta}_1$ and $\dot{\theta}_2$ denote the corresponding angular velocities. The system state vector is defined as:

$$x = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^\top \tag{17}$$

The system is controlled through two torque inputs T_1 and T_2. The experimental parameters are set as:

$$m = 0.5 \quad L = 0.5 \quad c = 0 \quad g = 1.$$

The controller is updated with a fixed discretization step size $\Delta t = 0.02$.

In this experiment, the verification objective is to check whether the system always satisfies the prescribed state constraints within a given time horizon. Specifically, for the time interval $t \in [0,\ 0.4]$, the system states are required to satisfy:

$$x(t) \in [-1.5,\ 1.5]^4$$

Under the discrete-time setting, this specification is equivalent to ensuring that the system states remain within the prescribed safe region for all control steps $k \in [0,\ 20]$.

[Airplane Control System Lopez et al. (2025)]. The airplane control benchmark describes a simplified fixed-wing aircraft dynamics model and is mainly used to evaluate the safety of neural network control systems in high-dimensional continuous state spaces. The system states include position, velocity, attitude angles, and angular velocities, defined as:

$$x = [s_x, s_y, s_z, v_x, v_y, v_z, \phi, \theta, \psi, r, p, q]^\top \tag{18}$$

where (s_x, s_y, s_z) denote the position of the aircraft center of mass, (v_x, v_y, v_z) are the velocity components, (ϕ, θ, ψ) represent the roll, pitch, and yaw angles, and (p, q, r) denote the body-frame angular velocities.

The system dynamics consist of translational and rotational motions. The control inputs include forces and torques along three axes. For simplicity, aerodynamic effects are incorporated into the control force terms. The experimental parameters are configured as:

$$m = 1 \quad I_x = I_y = I_z = 1 \quad I_{xz} = 0 \quad g = 1.$$

The controller is updated by a fixed sampling period $\Delta t = 0.1$.

The initial state set is defined as:

$$s_x = s_y = s_z = r = p = q = 0,$$

while the remaining state variables satisfy:

$$[v_x, v_y, v_z, \phi, \theta, \psi] \in [0.0,\ 1.0]^6.$$

This experiment evaluates whether the system violates safety constraints within the specified time horizon. Specifically, for the time interval $t \in [0,\ 2]$, the lateral position and attitude angles are required to satisfy:

$$s_y \in [-1,\ 1], \quad (\phi, \theta, \psi) \in [-1.0,\ 1.0]^3.$$

Under the discrete-time formulation, this specification corresponds to satisfying the above constraints for all control steps $k \in [0,\ 20]$.

[Adaptive Cruise Control (ACC) Lopez et al. (2025)]. The Adaptive Cruise Control (ACC) system models a typical longitudinal vehicle-following control problem, where the objective is to maintain a safe following distance while allowing the ego vehicle to smoothly track the leading vehicle. This benchmark is widely used to evaluate the ability of control strategies to preserve safety constraints in continuous dynamical systems.

The system consists of a leading vehicle and an ego vehicle. The leading vehicle is treated as part of the external environment, while the ego vehicle adjusts its speed through acceleration control. The system state vector is defined as:

$$x = [d, v_e, v_l]^\top, \tag{19}$$

where d denotes the relative distance between the two vehicles, v_e is the velocity of the ego vehicle, and v_l is the velocity of the leading vehicle. The control input is the longitudinal acceleration a of the ego vehicle.

The system dynamics are governed by the relative velocity between the vehicles. The ego-vehicle velocity is updated according to the control acceleration, while the leading-vehicle velocity varies within a prescribed range. The controller update step size is set to $\Delta t = 0.1$.

The initial state set is specified as:

$$d \in [10,\ 12], \quad v_e \in [10,\ 11], \quad v_l \in [10,\ 11].$$

The verification objective is to determine whether the system always satisfies the safety constraints within the given time horizon. Specifically, during the interval $t \in [0,\ 5]$, the relative distance is required to satisfy:

$$d(t) \geq 2.$$

Under the discrete-time setting, this specification corresponds to ensuring that the relative distance remains above the prescribed threshold for all control steps $k \in [0,\ 50]$.

[Single Pendulum System Lopez et al. (2025)]. The single pendulum system is a classical nonlinear control benchmark commonly used to evaluate the stability and safety of control systems under low-dimensional continuous dynamics. The system consists of a rigid pendulum rod of length L with a point mass m attached at its end, moving under gravitational effects.

Let θ denote the angular deviation from the vertical direction and let $\dot{\theta}$ denote the corresponding angular velocity. The system state vector is defined as:

$$x = [\theta, \dot{\theta}]^\top. \tag{20}$$

The system is regulated through a control torque input u to maintain operation near the target equilibrium point.

The experimental parameters are configured as:

$$m = 0.5 \quad L = 0.5 \quad g = 1.$$

The controller is discretized with a fixed update step size $\Delta t = 0.02$. The initial state set is specified as:

$$\theta \in [1.0,\ 1.3], \quad \dot{\theta} \in [1.0,\ 1.3].$$

The verification objective is to determine whether the system states always satisfy the prescribed safety constraints within the given time interval. Specifically, for $t \in [0,\ 1]$, the system states are required to satisfy:

$$\theta(t) \in [-1.5,\ 1.5], \quad \dot{\theta}(t) \in [-1.5,\ 1.5].$$

Under the discrete-time setting, this specification is equivalent to maintaining the states within the prescribed safe region for all control steps $k \in [0,\ 50]$.

[Ground Robot Rober et al. (2022)]. We consider the linearized Ground Robot model from Rober et al. (2022):

$$x_{t+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u_t \tag{21}$$

In this model, let $x_t = [p_x, p_y]^\top$ denote the robot position in the two-dimensional x-y plane, and $u_t = [v_x, v_y]^\top$ denote the control input (velocity). The initial state space is defined as $[-5.5, -4.5] \times [-0.5, 0.5]$, while the unsafe region is specified as $[-1, 1] \times [-1, 1]$. The sampling time is set to $t_s = 1$s. The verification objective is to guarantee that the system does not enter unsafe region within 9 time steps.

In addition, Rober et al. (2023b) replaces the Ground Robot NN controller with a defect-injected controller trained to deviate along the direction $y = -x$. In this chapter, we refer to this benchmark as *Ground Robot buggy*. The verification objective of this experiment is to determine the system safety within 6 time steps. Due to the injected defect, the ground-truth verification result of this benchmark is *Unsafe*.

A.4.7 Compared Verification Approaches

In experiment section, we conduct a series of experiments to evaluate *SynVAR*. We compare against representative verification tools based on solely forward reachability analysis, including *Reach-LP-GSG* Everett et al. (2020), *ReReach-LP* Rober et al. (2022), *ReachMM-CG* Harapanahalli et al. (2023), *ReachLibBaB* Entesari et al. (2023), and solely backward reachability analysis, including *BReach-LP* Rober et al. (2023b), *DRIP-Hpoly* Everett et al. (2023), *HyBReach-LP* Rober et al. (2023a), and state-of-the-art tool *NNV2.0* Lopez et al. (2023), *CROWN-Reach* Manzanas Lopez et al. (2024), *CORA* Kochdumper et al. (2023), and *JuliaReach* Bogomolov et al. (2019). We will outline the main principles behind these comparison tools below.

- **Reach-LP-GSG** Everett et al. (2020): Reach-LP-GSG is a forward reachability-based greedy simulation-guided partitioning method. It leverages simulation trajectories to identify regions that significantly influence boundary expansion, and prioritizes partitioning of the corresponding subspaces, making it more targeted than uniform partitioning.
- **ReReach-LP** Rober (2023): ReReach-LP is a forward method based on direct computation and iterative refinement of N-step reachable sets. Compared to step-by-step propagation, it mitigates error accumulation by directly tightening multi-step reachable sets, typically achieving higher verification precision at the cost of increased computational overhead.
- **ReachMM-CG** Harapanahalli et al. (2023): ReachMM-CG is a forward reachability analysis method that combines mixed-monotonicity decomposition with simulation-guided greedy partitioning. It leverages mixed-monotone system decomposition to obtain tighter interval over-approximations of reachable sets, and uses simulation feedback to prioritize refinement of subregions that contribute most to over-approximation error.
- **ReachLibBaB** Entesari et al. (2023): ReachLibBaB is a BaB-based reachability verification method built on the ReachLib framework. It integrates neural network bound propagation with systematic input domain partitioning, and iteratively refines the reachable set estimation by splitting critical regions.
- **BReach-LP** Rober et al. (2022): BReach-LP is a basic backward reachability analysis method based on box representations and linear programming. Starting from the unsafe or target region, it iteratively computes predecessor sets, allowing more direct utilization of unsafe constraints.
- **DRIP-Hpoly** Everett et al. (2023): DRIP-Hpoly is a backward reachability analysis method based on polytope representations. It uses more expressive polytopes to characterize backward reachable sets and iteratively tightens the bounds, offering finer geometric representations than box-based methods, albeit with higher computational complexity.
- **HyBReach-LP** Rober et al. (2023b): HyBReach-LP is a hybrid backward analysis method that combines target set partitioning with backward reachable set refinement. It performs iterative refinement on both the target region and intermediate backward reachable sets during backtracking, improving verification precision at the expense of increased computational cost.
- **NNV 2.0 Lopez et al. (2023)**: NNV 2.0 performs exact or conservative reachability analysis for closed-loop systems based on Star sets, and supports various NN architectures as well as continuous and discrete dynamical models.

- **CROWN-Reach Manzanas Lopez et al. (2024)**: CROWN-Reach combines the CROWN of neural network bound propagation methods with dynamic system reachability analysis, and employs symbolic interval propagation together with Taylor models to efficiently verify closed-loop systems with neural network controllers.
- **CORA Kochdumper et al. (2023)**: CORA is a verification toolbox for continuous and hybrid-system reachability analysis. It adopts set representations such as zonotopes and polynomial zonotopes to overapproximate systems with neural network controllers, while supporting nonlinear system dynamics.
- **JuliaReach Bogomolov et al. (2019)**: JuliaReach is a Julia-based reachability analysis framework that uses Taylor models to propagate controlled system dynamics and applies zonotope-based overapproximation for NNs, enabling formal analysis of closed-loop systems with NN controllers.

A.5 Future Work

Although the proposed method achieves promising improvements in both verification capability and computational efficiency, several important challenges remain for more complex real-world systems. Future research directions can be explored from the following aspects.

- **Scalable Verification for Large-Scale Networks and Long-Horizon Closed-Loop Systems.** As controller sizes, system dimensions, and closed-loop horizons continue to increase, verification methods relying solely on space partitioning and iterative refinement may suffer from rapid branch explosion and substantially increased propagation costs. Future work may further improve scalability by developing tighter linear relaxations, more efficient bound propagation techniques, and adaptive computational budget allocation strategies. In addition, parallel computing and hardware acceleration can be incorporated to improve the efficiency of large-scale verification tasks.
- **Verification of Closed-Loop Systems Under Uncertainty and Stochastic Effects.** Real-world systems are often affected by model uncertainties, external disturbances, and sensor noise, while some scenarios additionally exhibit stochastic behaviors. For such systems, uncertainty sets and disturbance models can be explicitly incorporated into the reachability analysis framework to provide robust safety guarantees. For stochastic systems, we further investigate probabilistic reachability analysis, probabilistic safety verification, and confidence evaluation methods, which can also be combined with sample-driven falsification approaches to obtain safety assurance results that better satisfy practical engineering requirements under limited computational budgets.

In practical applications, modern control systems are often composed of neural network modules, symbolic logic components, and optimization-based controllers. Future work may integrate the proposed reachability verification framework with control barrier functions and runtime monitoring techniques to construct a unified safety architecture combining offline formally verified guarantees with online runtime protection. Moreover, unified modeling and verification interfaces for neuro-symbolic systems can be further explored, enabling the flexibility of learning-based components to complement the interpretability and provability of symbolic methods, thereby supporting deployment in larger-scale and higher-risk safety-critical applications.