Scalable Synthesis of Formally Verified Neural Value Function for Hamilton-Jacobi Reachability Analysis

YUJIE YANG, Tsinghua University, China and Carnegie Mellon University, USA
HANJIANG HU, Carnegie Mellon University, USA
TIANHAO WEI, Carnegie Mellon University, USA
SHENGBO EBEN LI, Tsinghua University, China
CHANGLIU LIU*, Carnegie Mellon University, USA

Hamilton-Jacobi (HJ) reachability analysis provides a formal method for guaranteeing safety in constrained control problems. It synthesizes a value function to represent a long-term safe set called feasible region. Early synthesis methods based on state space discretization cannot scale to high-dimensional problems, while recent methods that use neural networks to approximate value functions result in unverifiable feasible regions. To achieve both scalability and verifiability, we propose a framework for synthesizing verified neural value functions for HJ reachability analysis. Our framework consists of three stages: pre-training, adversarial training, and verification-guided training. We design three techniques to address three challenges to improve scalability respectively: boundary-guided backtracking (BGB) to improve counterexample search efficiency, entering state regularization (ESR) to enlarge feasible region, and activation pattern alignment (APA) to accelerate neural network verification. We also provide a neural safety certificate synthesis and verification benchmark called Cersyve-9, which includes nine commonly used safe control tasks and supplements existing neural network verification benchmarks. Our framework successfully synthesizes verified neural value functions on all tasks, and our proposed three techniques exhibit superior scalability and efficiency compared with existing methods.

JAIR Associate Editor: Quanquan Gu

JAIR Reference Format:
Yujie Yang, Hanjiang Hu, Tianhao Wei, Shengbo Eben Li, and Changliu Liu. 2025. Scalable Synthesis of Formally Verified Neural Value Function for Hamilton-Jacobi Reachability Analysis. *Journal of Artificial Intelligence Research* 83, Article 19 (July 2025), 36 pages. DOI: 10.1613/jair.1.16946

1 Introduction

Safety is a primary concern in controller design, especially for control systems interacting with the physical world, such as autonomous driving and robot locomotion. In these systems, safety is usually specified by inequality constraints on system states. For example, safety constraints in a robot locomotion task require the distance between the robot and surrounding obstacles to be always positive. Such safety constraints must be satisfied not only in a single time step but also in all time steps over an infinite horizon. When designing a controller, it is important to know from which states it can satisfy the infinite-horizon safety constraints and from which states

[*] Corresponding Author.

Authors' Contact Information: Yujie Yang, ORCID: 0000-0001-7222-0019, yangyj21@mails.tsinghua.edu.cn, Tsinghua University, Beijing, China and Carnegie Mellon University, Pittsburgh, PA, USA; Hanjiang Hu, ORCID: 0000-0002-5698-5887, hanjianh@andrew.cmu.edu, Carnegie Mellon University, Pittsburgh, PA, USA; Tianhao Wei, ORCID: 0000-0003-2505-4585, twei2@andrew.cmu.edu, Carnegie Mellon University, Pittsburgh, PA, USA; Shengbo Eben Li, ORCID: 0000-0003-4923-3633, lishbo@tsinghua.edu.cn, Tsinghua University, Beijing, China; Changliu Liu, ORCID: 0000-0002-3767-5517, cliu6@andrew.cmu.edu, Carnegie Mellon University, Pittsburgh, PA, USA.

DOI: 10.1613/jair.1.16946

it cannot. The deployment of any controller should be restricted to a set of states where long-term constraint satisfaction is ensured, which is called a *feasible region*.

Hamilton-Jacobi (HJ) reachability analysis provides a formal method for computing feasible regions of control systems with safety constraints [5]. In HJ reachability analysis, a feasible region is represented by the zero-sublevel set of a value function, which is defined as the maximum value of the constraint function over a trajectory. In general nonlinear systems, exactly computing the value function is difficult because it involves solving the HJ partial differential equation, which does not have a closed-form solution in most cases [5]. Traditional methods numerically solve the HJ equation on a grid representing a discretization of the state space [28, 29]. These methods' computational complexity grows exponentially with state dimension, making them intractable in high-dimensional systems. Although some techniques, such as system decomposition [11], are proposed for accelerating value function computation, they only apply to some special scenarios. Recent methods use neural networks to approximate the solution to the HJ equation by minimizing the error between the two sides of the equation [16, 6]. The error is computed on states randomly sampled in the state space and minimized by gradient-based optimization algorithms. Although these methods scale well to high-dimensional systems, the zero-sublevel sets of their value functions are no longer guaranteed to be valid feasible regions due to neural network approximation errors. Specifically, their zero-sublevel sets may violate two basic properties of a feasible region: *constraint satisfaction* and *forward invariance*. Constraint satisfaction means all states in a feasible region satisfy the safety constraint themselves. Forward invariance means starting from any state in a feasible region, its subsequent states can always be kept in this region by some control policy. Violating these two properties may cause possible constraint violations, even starting from a state inside the zero-sublevel set, making the value function unreliable for safe control.

The problem of invalid feasible regions necessitates verification of neural HJ reachability value functions. Recently, some researchers have begun to use neural network verification tools to formally verify and synthesize neural safety certificates, such as neural control barrier functions (CBFs) [37, 43] and neural control Lyapunov functions (CLFs) [10, 2]. Similar to HJ reachability value function, these safety certificates also represent feasible regions by their zero-sublevel sets. The difference is that these functions are not defined through equations but through certain inequality conditions. For example, the time derivative of a CBF must be upper bounded by an extended class $\mathcal{K}$ function, and a CLF must be a positive definite function with negative time derivatives. Existing works try to verify whether these conditions are strictly satisfied by the neural safety certificates in the entire state space. Such problems can be transformed into standard neural network verification problems, which can be solved by existing verification tools [25]. This verification procedure can also be embedded into neural safety certificate synthesis, resulting in verification-guided training methods [37, 43]. If verification fails on a synthesized safety certificate, the found counterexamples are added to the dataset, and the safety certificate is further trained on these counterexamples. This process is repeated until the neural safety certificate is successfully verified. However, these methods currently only work on low-dimensional or linear systems and are difficult to scale to high-dimensional nonlinear systems and real-world control tasks. Through our study, we discover three main challenges that restrict the scalability of these methods:

- **Difficulty of searching and eliminating counterexamples.** Successful verification requires strict satisfaction of inequality conditions in all states, and not a single counterexample is allowed. However, counterexamples become extremely sparse in high-dimensional spaces, making it difficult to find and eliminate all of them.
- **Severe shrinkage of feasible region.** When training neural safety certificates on counterexamples, their zero-sublevel sets tend to shrink so that the inequality conditions can be more easily satisfied. Although slight shrinkage is sometimes acceptable, severe shrinkage can be a serious problem because it results in

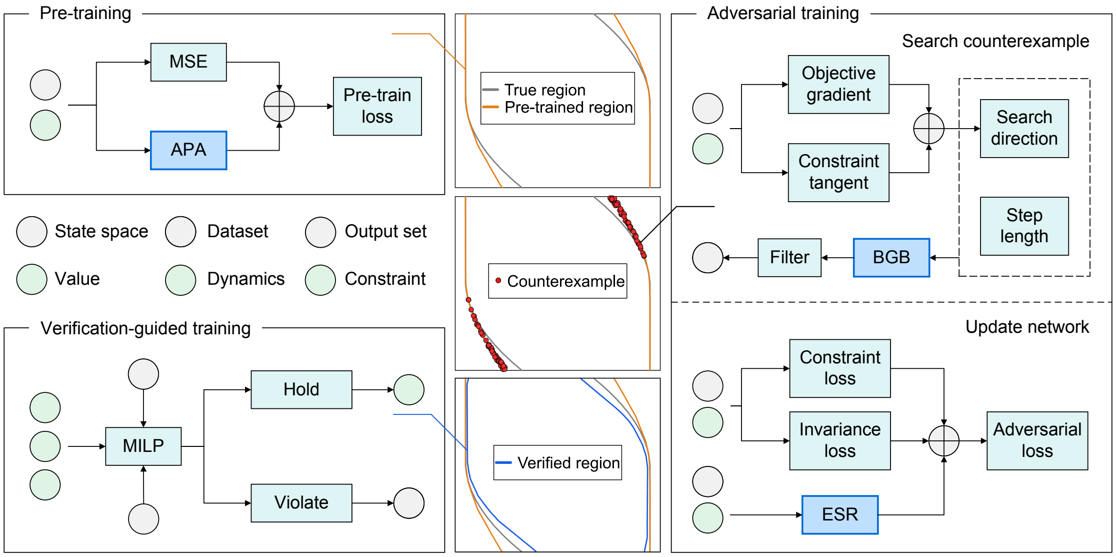


Fig. 1. Neural HJ reachability value function synthesis framework. Our framework consists of three stages: pre-training, adversarial training, and verification-guided training. The circles of state space and dataset in pre-training and adversarial training mean randomly sampling states from these sets, while in verification-guided training, the state space and output set are in analytical form. Our three key techniques are highlighted in blue boxes, namely APA, BGB, and ESR. The three middle figures show the synthesis results on a 2D task Double Integrator. The pre-trained feasible region is larger than the true feasible region, indicating that it is invalid. After adversarial training and verification-guided training, the feasible region becomes valid. Details of the experiment can be found in Section 5.

overly conservative control policies and poor control performance. We find in our experiments that, in some cases, the feasible region shrinks so much that it even disappears.

- **High computational complexity of verification.** The computational complexity of neural network verification algorithms typically grows exponentially with the input dimension, which equals the system's state dimension. As shown in our experiments, verifying a relatively small value network in a 4-dimensional system can take more than 2 hours on a common computing platform.

The above analysis reveals a key challenge of synthesizing neural HJ reachability value functions: the trade-off between scalability and verifiability. Traditional numerical methods based on state space discretization ensure verifiability but sacrifice scalability. Neural HJ reachability methods scale well to high-dimensional systems, but their synthesized networks are not verified. Recent verification-guided training methods provide a promising way to synthesize verifiable neural safety certificates, but they again sacrifice scalability because of the aforementioned three challenges. To achieve both scalability and verifiability, we propose a scalable framework for synthesizing formally verified neural HJ reachability value functions, as shown in Figure 1. Our framework consists of three stages: pre-training, adversarial training, and verification-guided training. Pre-training and verification-guided training are widely used in existing neural safety certificate synthesis methods [37, 10, 2]. The former aims to obtain a reasonable approximation of the value function, while the latter aims to fine-tune the network on counterexamples until it becomes a valid safety certificate. However, directly performing verification on a pre-trained value network is inefficient because a pre-trained network usually has a large number of counterexamples, while only a single one can be found in each verification step. To improve fine-tuning efficiency, we add an adversarial training stage between them, which searches and eliminates counterexamples in a batched manner. Verification-guided training does not start until adversarial training can hardly find any counterexamples. Although there are many advanced methods for neural network verification, e.g., α, β-CROWN [40, 36], we choose a basic method: mixed integer linear programming (MILP) [34]. This is because most advanced methods are designed for robust image classification problems, while we consider safety certificate synthesis problems, which have much lower input dimensions and smaller neural networks. In such small-scale problems, those advanced methods perform even worse than the basic MILP [36]. Note that this does not mean that our problem is simpler than robust image classification; in some sense, our problem is even harder as we require certain properties to hold in the entire input space instead of a small disturbance set. With our three-stage framework in place, the next step is to find specific methods to solve the problem in each stage. Although there are many well-studied methods for these three stages, we find that directly using state-of-the-art methods makes the framework quickly fail as the problem dimension increases, because these methods cannot effectively solve the aforementioned three challenges. To this end, we propose three techniques to improve the scalability of our framework, each designed to address one of the three challenges. First, we find that counterexample search in adversarial training is difficult because existing gradient-based search methods are inefficient in searching along the boundary of feasible region. We propose a backtracking method that rotates the search direction towards the boundary to accelerate counterexample search. Second, we exploit the fact that constraint-satisfying states that enter the feasible region in one step are also feasible and penalize the value of these states in fine-tuning to alleviate the shrinkage of feasible region. Third, we discover that the number of linear segments of the neural network greatly affects the computational complexity of solving MILP. We design a regularization term for network pre-training to reduce the number of linear segments and thus accelerate verification. Our main contributions are summarized as follows.

- We propose a scalable framework for synthesizing formally verified neural HJ reachability value functions. Our framework synthesizes neural value functions from coarse to fine with high efficiency through three stages: pre-training, adversarial training, and verification-guided training. Pre-training approximates the solution to an HJ equation by gradient descent on data samples and obtains a value network that is likely to be invalid. Adversarial training searches counterexamples in a batched manner based on necessary and sufficient conditions for feasible region and fine-tunes the value network to eliminate counterexamples. Verification-guided training formulates the value network verification problem as an MILP and further fine-tunes the network on counterexamples until it is verified.
- To accelerate counterexample search in adversarial training, we propose an algorithm called boundary-guided backtracking (BGB) that efficiently searches along the boundary of feasible region. When approaching the boundary, BGB rotates the search direction towards the tangent plane of the boundary so that larger step sizes can be taken without stepping out of the feasible region.
- To alleviate feasible region shrinkage, we present entering state regularization (ESR) that adds a penalty term to the loss function when fine-tuning the value network. ESR first identifies constraint-satisfying states that enter the feasible region in one step and then encourages the values of these states to be negative so that they are included in the feasible region.
- To accelerate MILP-based verification, we design a regularization method called activation pattern alignment (APA) for pre-training of value network and dynamics network. APA reduces linear segments of a neural network by penalizing the difference in the activation patterns of neighboring states while minimizing the loss of network approximation ability.
- We provide a benchmark called Cersyve-9 for neural safety certificate synthesis and verification in safe control problems, which supplements existing neural network verification benchmarks. Cersyve-9 contains

nine commonly used control tasks with various dimensions, including linear and nonlinear system dynamics and safety constraints. Extensive experiments on Cersyve-9 demonstrate the effectiveness, scalability, and efficiency of our synthesis framework. The code of our benchmark is available on GitHub[1].

2 Related Works

In this section, we review existing works on the synthesis of neural HJ reachability value functions and the verification of neural safety certificates.

2.1 Synthesis of Neural HJ Reachability Value Function

Traditional HJ reachability analysis computes the value function by numerically solving the HJ PDE on a discretized grid of the state space. The computational complexity of this method grows exponentially with state dimension, making it inapplicable to high-dimensional problems. To deal with this issue, researchers have explored using neural networks to approximate the value function.

A straightforward method for learning a neural value function is to minimize the error between the two sides of the HJ equation by gradient descent on state samples [14, 6]. However, this method is hard to converge because the HJ equation does not yield a contraction mapping and thus does not satisfy the convergence conditions of fixed point iteration. In practice, this method typically requires specific initialization of the value function or relies on certain neural network architectures to converge to the correct solution. Fisac et al. [16] solve this problem by introducing a discount factor into the value function, modifying the original maximum constraint formulation of HJ reachability to a maximum discounted constraint formulation. Under the discounted formulation, the HJ equation also changes to a discounted version, which yields a contraction mapping and enables convergence of fixed point iteration with an arbitrary initialization. This makes temporal difference learning methods in reinforcement learning (RL) applicable to computing the value function. Since then, the discounted value function has been extensively used for neural HJ reachability analysis in safe control tasks, especially when combined with RL algorithms. For example, Hsu et al. [20] consider reach-avoid problems and add goal information to the value function proposed by Fisac et al. [16]. They derive a discounted reach-avoid Bellman backup and prove that their reach-avoid Q-learning algorithm converges to an arbitrarily tight conservative approximation of the reach-avoid set. Yu et al. [41] establish a self-consistency condition for computing the value function of a specific policy. They use the value function as the objective function and constraint for shield and main policies, respectively. The value function is also used for policy switching during training and safety shield during evaluation. He et al. [18] applies the method proposed by Hsu et al. [20] to train a reach-avoid value function in a quadrupedal robot locomotion task. Their value function controls the switch between an agile policy and a recovery policy, and also guides the recovery policy as an objective function.

Despite these advancements, there is an inherent problem in approximating value function with neural networks: the zero-sublevel set of the value network may not be a valid feasible region due to approximation errors. Specifically, the zero-sublevel set may violate the two basic properties of a feasible region: constraint satisfaction and forward invariance. This can be problematic when using these value networks for constructing constraints in policy optimization or monitoring unsafe actions in safety filters. With these two properties unsatisfied, even if the current state is inside the zero-sublevel set, the system may still run into a constraint-violating state sometime in the future. This problem necessitates additional verification of the value network, which is not addressed by existing works.

[1] https://github.com/intelligent-control-lab/Cersyve.jl

2.2 Verification of Neural Safety Certificates

Safety certificates are real-valued functions of system state that are used to represent feasible regions and construct constraints or safety filters of control policy. HJ reachability value function is a kind of safety certificate, and two other representative examples are CBF and CLF. CBF and CLF are defined through certain inequality conditions which, when strictly satisfied, ensure that the zero-sublevel sets of these safety certificates are feasible regions. Similar to HJ reachability value function, CBF and CLF can also be represented by neural networks, and the resulting neural CBF and CLF also face the problem of verification.

With the development of neural network verification tools [25], some recent studies have begun to formally verify the inequality conditions of neural CBF and CLF. For example, Zhang et al. [43] first decompose a neural CBF into piecewise linear segments and then solve a nonlinear program to verify the safety of each segment. To deal with the non-differentiable ReLU activation function, they leverage a generalization of Nagumo's theorem to prove invariance of sets with non-smooth boundaries and derive necessary and sufficient conditions for safety. While Zhang et al. [43] focus on verifying a given neural CBF, verification of neural safety certificates can also be combined with their training process. This yields a verification-guided training scheme of neural safety certificates, which iterates between a learner and a verifier. The learner updates the certificate on data samples to enforce the satisfaction of safety properties. The verifier either verifies the certificate's validity in the entire state space or generates counterexamples and adds them to the dataset for further training. This iterative procedure terminates when no counterexample is found by the verifier, in which case the neural safety certificate is formally verified. This training scheme is widely used for synthesizing formally verified neural CBFs [32, 1, 12] and neural CLFs [10, 2, 13]. To improve the efficiency of the verifier, Wang et al. [37] leverage the Branch-and-Bound scheme to identify partitions of the state space that are not guaranteed to satisfy CBF conditions. Additional data from these partitions are incorporated into the training dataset for further optimization. To accelerate neural CBF training, some works exploit the Lipschitz continuity property of neural CBF and use robust training techniques to ensure the satisfaction of CBF conditions [4, 33].

Despite these exploratory works, challenges remain in scaling verifiable neural safety certificate synthesis methods to high-dimensional problems. Most existing methods only apply to control systems with less than four state dimensions [2, 37, 33] or special systems with four to eight dimensions whose state consists of the derivatives of the same variable, and the dynamics is described by a single scalar ordinary differential equation [32, 1]. Chang et al. [10] and Dai et al. [13] synthesize verified neural Lyapunov functions on six-dimensional humanoid and quadrotor systems respectively, but their training takes hours, and the obtained feasible regions are small, which may result in overly conservative control policies. Through our study, we discover three main challenges that restrict the scalability of these methods: 1) difficulty of searching and eliminating counterexamples, 2) severe shrinkage of feasible region, and 3) high computational complexity of verification. We propose three techniques to mitigate these three challenges respectively, and they together significantly improve the scalability of our synthesis framework.

3 Preliminaries

This section introduces some basic concepts in safe control, HJ reachability analysis, and neural network verification, and formalizes the neural HJ reachability synthesis problem.

3.1 State Constraint and Feasible Region

Consider a discrete-time deterministic control system:

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

where $x \in \mathcal{X}$ is the state, $u \in \mathcal{U}$ is the control input, f is the system dynamics, and $t \in \mathbb{N}$ is the time step. A control policy maps a state to a control input, i.e., $\pi : \mathcal{X} \to \mathcal{U}$. The closed-loop dynamics under control policy π is denoted as $f_\pi(x) \coloneqq f(x, \pi(x))$. Safety constraint of the system is specified by:

$$h(x_t) \leq 0, \forall t \in \mathbb{N}, \tag{2}$$

where h is the constraint function and the inequalities (2) are called state constraints. Before implementing a control policy, it is necessary to identify the states where the closed-loop system always satisfies the state constraints. Such states constitute the feasible region of the policy, which is defined as follows.

Definition 1 (Feasible region). *A feasible region of policy π, denoted as X^π, is a subset of the state space $\mathcal{X}$ such that $\forall x \in \mathrm{X}^\pi$, $h(x_t) \leq 0, t \in \mathbb{N}$, where $x_0 = x$ and $x_{t+1} = f_\pi(x_t)$.*

From the above definition, we can derive a set of necessary and sufficient conditions for feasible region: constraint satisfaction and forward invariance.

Theorem 1 (necessary and sufficient conditions for feasible region). *X^π is a feasible region of π if and only if*

(1) *(Constraint satisfaction)* $\forall x \in \mathrm{X}^\pi, h(x) \leq 0$.
(2) *(Forward invariance)* $\forall x \in \mathrm{X}^\pi, f_\pi(x) \in \mathrm{X}^\pi$.

The proof of Theorem 1 follows directly from Definition 1 and is omitted here. Constraint satisfaction means all states in a feasible region satisfy the state constraint at the current time step. Forward invariance means that for any state in the feasible region, its next state under the closed-loop dynamics still lies in this region. These two conditions are useful for determining and identifying feasible regions because they only involve a single-step state transition instead of infinite steps as Definition 1. They are also used as the conditions for verifying neural HJ reachability value functions in this paper.

3.2 Hamilton-Jacobi Reachability Analysis

Hamilton-Jacobi (HJ) reachability analysis identifies the feasible region of a control system with state constraints by computing a value function. In a closed-loop system under a given control policy, the value function is defined as the maximum value of the constraint function in a trajectory sampled by the policy.

Definition 2 (HJ reachability value function). *The HJ reachability value function of control policy π is defined as*

$$V^\pi(x) \coloneqq \max_{t \in \mathbb{N}} h(x_t), \tag{3}$$

where $x_0 = x$ and $x_{t+1} = f_\pi(x_t)$.

A desirable property of the value function is that its zero-sublevel set is a feasible region.

Theorem 2. *The zero-sublevel set of V^π, denoted as $\mathrm{X}_{V^\pi} \coloneqq \{x \in \mathcal{X} | V^\pi(x) \leq 0\}$, is a feasible region of π.*

Proof. We prove that X_{V^π} satisfies the two conditions in Theorem 1. $\forall x \in \mathrm{X}_{V^\pi}$, we have $h(x) \leq \max_{t \in \mathbb{N}} h(x_t) = V^\pi(x) \leq 0$. Thus, X_{V^π} satisfies constraint satisfaction. $\forall x \in \mathrm{X}_{V^\pi}$, we have $V^\pi(f_\pi(x)) = \max_{t \geq 1} h(x_t) \leq \max_{t \in \mathbb{N}} h(x_t) = V^\pi(x) \leq 0$. Thus, X_{V^π} satisfies forward invariance. Therefore, X_{V^π} is a feasible region of π. □

Theorem 2 makes the value function useful for representing feasible regions and synthesizing safe controllers. For example, with a possibly unsafe nominal policy and a safe backup policy π, one can determine whether the nominal policy will lead to a possibly unsafe state by checking if the next state is in X_{V^π}. Specifically, if the next state is in X_{V^π}, it is safe as π can keep the state always in X_{V^π}. If the next state is not in X_{V^π}, it is possibly unsafe as π has no safety guarantee outside X_{V^π}, and we should replace the nominal policy with π to compute a safe action. This zero-sublevel set property is one of the most fundamental properties of safety certificates. In addition to HJ reachability value function, other safety certificates, such as CBF and CLF, also have similar properties.

In a stochastic control system, the HJ reachability value function satisfies an HJ PDE [5]. In a deterministic closed-loop system, the minimum and maximum operators on control inputs and disturbances in the HJ PDE can be omitted, resulting in a simplified equation called the risky self-consistency condition.

Theorem 3 (Risky self-consistency condition). *Let V^π be the value function of π, $\forall x \in \mathcal{X}$, we have*

$$V^\pi(x) = \max\{h(x), V^\pi(f_\pi(x))\}. \tag{4}$$

The risky self-consistency condition is a recursive relationship between the values of the previous and subsequent states. In RL, the value function of reward also has a similar self-consistency condition. Here, the name “risky" distinguishes the HJ reachability value function from the reward value function, reflecting the former’s relationship to safety constraints. With the risky self-consistency condition, we can compute the value function by solving Equation (4). However, this equation does not have an analytical solution in most cases, and traditional numerical methods based on state space discretization cannot scale to high-dimensional systems. This necessitates using a neural network to represent the value function and approximate the solution to Equation (4).

3.3 Synthesizing Neural HJ Reachability Value Function

A straightforward method for fitting a neural network to the solution to Equation (4) is to minimize the error between the two sides of the equation. However, this method is hard to converge because Equation (4) does not yield a contraction mapping. To create a contraction mapping, Fisac et al. [16] modify the risky self-consistency condition to a discounted version:

$$V^\pi(x) = (1-\gamma)h(x) + \gamma\max\{h(x), V^\pi(f_\pi(x))\}, \tag{5}$$

where $\gamma \in (0, 1)$ is a discounted factor. They prove that Equation (5) induces a contraction mapping on the value function space under the infinity norm [16]. This ensures that a fixed point iteration converges to the unique solution to Equation (5), which is a discounted version of the value function. As the discount factor γ approaches one, the solution to (5) approaches that of (4), which is the original value function.

Suppose we use a feedforward neural network V_θ^π to represent the value function, where θ is the network parameters. To approximate the solution to Equation (5), we minimize the mean squared error (MSE) between the two sides of the equation:

$$L_{\text{RSC}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(V_\theta^\pi(x^{(i)}) - ((1-\gamma)h(x^{(i)}) + \gamma\max\{h(x^{(i)}), V_\theta^\pi(f_\pi(x^{(i)}))\}))^2, \tag{6}$$

where the subscript “RSC" stands for risky self-consistency condition, N is the number of state samples, and $x^{(i)}$ stands for the i-th state sample. In practice, the states are uniformly sampled from the state space. However, a problem with this method is that the obtained value network may not be a valid safety certificate. Specifically, the zero-sublevel set of V_θ^π, denoted as $\mathrm{X}_{V_\theta^\pi}$, may not strictly satisfy the necessary and sufficient conditions for a feasible region given in Theorem 1. The cause for this invalidity is two-fold: 1) approximation errors of V_θ^π make it not exactly the solution to (5), and 2) when the discount factor γ is less than one, the zero-sublevel set of the solution to the discounted self-consistency condition (5) is an over-approximation of that of the solution to the original self-consistency condition (4) [3]. Since the zero-sublevel set of the solution to (4) is already the maximum feasible region of π, any of its over-approximations is not a valid feasible region. This invalidity may lead to unsafe behaviors when using V_θ^π for synthesizing control policies. To address this issue, this paper aims to synthesize neural HJ reachability value functions with verified zero-sublevel sets.

Problem 1. *Given a control system* (1) *and a control policy π, synthesize a neural HJ reachability value function V_θ^π, such that its zero-sublevel set is a verified feasible region of π, i.e., it strictly satisfies the conditions of constraint satisfaction*

$$\forall x \in \mathcal{X}, V_\theta^\pi(x) \leq 0 \Rightarrow h(x) \leq 0, \tag{7}$$

and forward invariance

$$\forall x \in \mathcal{X}, V_\theta^\pi(x) \leq 0 \Rightarrow V_\theta^\pi(f_\pi(x)) \leq 0. \tag{8}$$

Note that the constraint satisfaction and forward invariance conditions only ensure that the zero-sublevel set of V_θ^π is a feasible region but do not ensure that V_θ^π is an exact solution to (4). In fact, they are only necessary conditions for V_θ^π to be an exact solution. We choose to verify these two conditions because exact verification of (4), which is an equation involving a neural network, is almost impossible due to neural network approximation errors. In addition, since any safety certificate represents a feasible region by its level set, these two conditions can also be used to verify other safety certificates, such as CBF and CLF, with minor modifications.

3.4 Verifying Neural Network via Mixed Integer Linear Programming

Verification of a neural network is to check whether the network's output lies in a specific output set for all inputs in a given input set [25]. Mathematically, let $\mathcal{X}$ be the input set, $\mathcal{Y}$ be the output set, and $\mathrm{NN}(\cdot)$ be the neural network. A verification problem requires to check whether the following assertion holds:

$$\forall x \in \mathcal{X}, y = \mathrm{NN}(x) \in \mathcal{Y}. \tag{9}$$

In this paper, we consider the case where $\mathrm{NN}(\cdot)$ is a feedforward neural network with ReLU activation functions. In this case, $\mathrm{NN}(\cdot)$ is a piecewise linear function, and the equality $y = \mathrm{NN}(x)$ can be encoded as a set of linear and integer constraints [34]. Moreover, if the input set $\mathcal{X}$ and the complement of the output set $\mathcal{Y}$ can be expressed by a finite number of linear constraints, e.g., $\mathcal{X}$ and the complement of $\mathcal{Y}$ are polytopes, assertion (9) can be checked by solving a mixed integer linear programming (MILP):

$$\text{find } x, \quad \text{s.t. } x \in \mathcal{X}, y \notin \mathcal{Y}, y = \mathrm{NN}(x). \tag{10}$$

Problem (10) tries to find a counterexample in the input set such that the corresponding output of the neural network is not in the output set. If the problem is feasible, the property to verify is violated and a counterexample is found. If the problem is infeasible, the property holds. Note that problem (10) is a feasibility problem, i.e., an optimization problem without an objective function. It is also possible to include an objective function in (10), with common examples like maximum violation and minimum disturbance [25]. In this paper, since we only focus on whether the property holds or not and have no preference for counterexamples, we omit the objective function to simplify the problem.

4 Method

This section formally introduces our neural HJ reachability value function synthesis framework. We first provide an overview of our synthesis framework and then detail three key techniques that significantly improve its scalability.

4.1 Overview

Our framework consists of three stages: pre-training, adversarial training, and verification-guided training. Pre-training learns a value network without verification, which probably violates the feasible region conditions. Adversarial training efficiently searches for counterexamples and eliminates most of them by fine-tuning the network. Verification-guided training finds remaining counterexamples by solving the MILP and further fine-tunes the network until the feasible region conditions are verified. Pre-training is performed first and is separate from the other two stages. Adversarial training is performed next, and if no counterexamples are found in a certain number of iterations, verification-guided training starts. If a counterexample is found, we return to adversarial training until the next time verification-guided training is triggered. This process is repeated until verification succeeds, at which point we have synthesized a valid neural value function. Compared with most existing neural safety certificate synthesis methods that include pre-training and verification-guided training [37, 10, 2], our framework adds an adversarial training stage between them. This is because a pre-trained value network usually has a large number of counterexamples, while only a single one can be found in each verification step, making verification-guided training inefficient. In contrast, adversarial training searches counterexamples in a batched manner, making the search much less expensive than that in verification-guided training in terms of computation. First using adversarial training to reduce counterexamples to a small number and then performing verification-guided training greatly improves fine-tuning efficiency.

In pre-training, we learn a value network by minimizing the following loss function.

$$L_{\text{pre}}(\theta) = L_{\text{RSC}}(\theta) + L_{\text{APA}}(\theta), \tag{11}$$

where $L_{\text{APA}}(\theta)$ is a regularization term computed by activation pattern alignment (APA), which reduces the number of linear segments of a neural network to accelerate verification. This technique will be detailed in Section 4.2. The pre-trained value network is not verified and may violate the feasible region conditions. We fine-tune the network in the next two stages to make its zero-sublevel set a verified feasible region.

In adversarial training, we first search for counterexamples, i.e., states that violate the feasible region conditions. For simplicity of narration, we call the counterexamples of the constraint satisfaction condition the *constraint counterexamples* and those of the forward invariance condition the *invariance counterexamples*. To find these two kinds of counterexamples, we solve two corresponding optimization problems. The optimization problem for finding constraint counterexamples is

$$\max_{x \in \mathcal{X}} \ h(x), \quad \text{s.t. } V_\theta^\pi(x) \leq 0, \tag{12}$$

and that for finding invariance counterexamples is

$$\max_{x \in \mathcal{X}} \ V_\theta^\pi(f_\pi(x)), \quad \text{s.t. } V_\theta^\pi(x) \leq 0. \tag{13}$$

If the optimal value of (12) is greater than zero, the solution is a constraint counterexample. Similarly, if the optimal value of (13) is greater than zero, the solution is an invariance counterexample. Generally, problem (12) and (13) are non-convex in both their objective functions and constraints and are thus difficult to solve. However, to find counterexamples, we do not need to solve them exactly but only need to find feasible points with positive objective functions, i.e., $h(x) > 0$ and $V_\theta^\pi(x) \leq 0$ for constraint counterexamples and $V_\theta^\pi(f_\pi(x)) > 0$ and $V_\theta^\pi(x) \leq 0$ for invariance counterexamples. To achieve this goal efficiently, we propose a gradient-based search method called boundary-guided backtracking (BGB), which will be detailed in Section 4.3.

After obtaining counterexamples, we store them in a dataset for fine-tuning the value network. In each iteration, we randomly sample the two kinds of counterexamples from the dataset and minimize their corresponding loss functions computed according to the feasible region conditions. The loss function for constraint counterexamples is

$$L_{\text{con}}(\theta) = \frac{1}{N_{\text{con}}} \sum_{i=1}^{N_{\text{con}}} -V_\theta^\pi(x_{\text{con}}^{(i)}), \tag{14}$$

and that for invariance counterexamples is

$$L_{\text{inv}}(\theta) = \frac{1}{N_{\text{inv}}} \sum_{i=1}^{N_{\text{inv}}} V_\theta^\pi(f_\pi(x_{\text{inv}}^{(i)})) - V_\theta^\pi(x_{\text{inv}}^{(i)}), \tag{15}$$

where N_{con} and N_{inv} are the numbers of constraint and invariance counterexamples respectively, and $x_{\text{con}}^{(i)}$ and $x_{\text{inv}}^{(i)}$ stands for the i-th constraint counterexample and invariance counterexample, respectively. We discover in our experiments that directly minimizing these two loss functions will result in severe shrinkage of the value network's zero-sublevel set. A possible reason for the shrinkage is that minimizing (15) only decreases the difference between the values of the next state and the current state, but their respective values may increase instead. To mitigate this problem, we include an additional regularization term in the loss function of adversarial training:

$$L_{\text{adv}}(\theta) = L_{\text{con}}(\theta) + L_{\text{inv}}(\theta) + L_{\text{ESR}}(\theta), \tag{16}$$

where $L_{\text{ESR}}(\theta)$ is computed using entering state regularization (ESR), which will be detailed in Section 4.4.

In verification-guided training, we use mixed integer linear programming (MILP) to verify the feasible region conditions or find counterexamples. If the conditions are verified, we obtain a valid neural value function, and the algorithm ends. If counterexamples are found, we add them to the dataset for further fine-tuning. The core problem is how to formulate the verification of the feasible region conditions as MILPs. In a standard verification problem (9), there is only one function $\text{NN}(\cdot)$, while in our problem, verification of each condition involves two functions, and their corresponding inequalities have an implication relationship. To deal with this problem, we first concatenate the two functions into a single function with two outputs:

$$\text{M}_{\text{con}}(x) = \begin{bmatrix} V_\theta^\pi(x) \\ h(x) \end{bmatrix}, \quad \text{M}_{\text{inv}}(x) = \begin{bmatrix} V_\theta^\pi(x) \\ V_\theta^\pi(f_\pi(x)) \end{bmatrix}. \tag{17}$$

With this concatenation, the feasible region conditions can be naturally expressed by restricting the output of the concatenated function in the complement of the second quadrant in a two-dimensional space. Take the constraint satisfaction condition as an example, "$V_\theta^\pi(x) \le 0 \Rightarrow h(x) \le 0$" means the output of M_{con} should not be in the second quadrant. Thus, we can define an output set whose complement is the second quadrant so that the output constraint "$y \notin \mathcal{Y}$" can be expressed by linear inequality constraints. Therefore, verification of the constraint satisfaction condition can be formulated as

$$\text{find } x, \quad \text{s.t. } x \in \mathcal{X}, y = \text{M}_{\text{con}}(x), y_1 \le 0, y_2 \ge 0, \tag{18}$$

where y_i stands for the i-th element of y. Here, we assume that the state space $\mathcal{X}$ is a hyperrectangle, which is true in most cases. Then, constraint $x \in \mathcal{X}$ in problem (18) can be expressed by linear inequalities. Now, as long as M_{con} is piecewise linear, problem (18) is an MILP. Since V_θ^π is a piecewise linear neural network, M_{con} is piecewise linear when the constraint function h is piecewise linear. Here, we assume that h is piecewise linear, and the nonlinear case will be left for future work[2]. Verification of the forward invariance condition can be similarly formulated as

$$\text{find } x, \quad \text{s.t. } x \in \mathcal{X}, y = \text{M}_{\text{inv}}(x), y_1 \le 0, y_2 \ge 0. \tag{19}$$

If the dynamics f and the policy π are both piecewise linear functions, M_{inv} is piecewise linear. Similar to h, we assume that f and π are piecewise linear, and the nonlinear case will be left for future work. To show how problems (18) and (19) can be encoded as MILPs, we explicitly write them in a unified standard form of MILP as

[2] One possible solution is to approximate a nonlinear h using piecewise linear functions, e.g., Taylor model.

follows. Here, we view $\mathrm{M}_{\mathrm{con}}$ and $\mathrm{M}_{\mathrm{inv}}$ as a single feedforward neural network with ReLU activation functions.

$$\begin{aligned}
\text{find} \quad & x && \text{(20a)}\\
\text{s.t.} \quad & \mathcal{X}_l \leq x \leq \mathcal{X}_u, && \text{(20b)}\\
& z_0 = x, && \text{(20c)}\\
& \hat{z}_j = W_j z_{j-1} + b_j,\ j = 1, \ldots, l, && \text{(20d)}\\
& \text{if } \ \hat{l}_{j,k} \geq 0,\ z_{j,k} = \hat{z}_{j,k},\ j = 1, \ldots, l-1,\ k = 1, \ldots, d_j, && \text{(20e)}\\
& \text{if } \hat{u}_{j,k} \leq 0,\ z_{j,k} = 0, && \text{(20f)}\\
& \text{otherwise,} \ \ z_{j,k} \leq \hat{z}_{j,k}, && \text{(20g)}\\
& \qquad\qquad\quad z_{j,k} \geq 0, && \text{(20h)}\\
& \qquad\qquad\quad z_{j,k} \leq \hat{z}_{j,k} - \hat{l}_{j,k}(1 - \delta_{j,k}),\ \delta_{j,k} \in \{0, 1\}, && \text{(20i)}\\
& \qquad\qquad\quad z_{j,k} \leq \hat{u}_{j,k}\delta_{j,k}, && \text{(20j)}\\
& \hat{z}_l = y, && \text{(20k)}\\
& y_1 \leq 0,\ y_2 \geq 0, && \text{(20l)}
\end{aligned}$$

where $\mathcal{X}_l$ and $\mathcal{X}_u$ are lower and upper bounds of $\mathcal{X}$, the inequality signs in (20b) represent element-wise comparisons, $\hat{z}_j$ and z_j are the j-th layers of the neural network $\mathrm{M}_{\mathrm{con}}$ or $\mathrm{M}_{\mathrm{inv}}$ before and after activation, $\hat{z}_{j,k}$ and $z_{j,k}$ are the k-th elements in the i-th layers, $\hat{l}_{j,k}$ and $\hat{u}_{j,k}$ are the lower and upper pre-activation bounds, $\delta_{j,k}$ is the binary variable for activation status. Equations (20e) to (20j) are encodings of ReLU, which follow the method proposed by Tjeng et al. [34]. The pre-activation bounds $\hat{l}_{j,k}$ and $\hat{u}_{j,k}$ can be computed by any reachability-based verification method [25], and we use CROWN [42] in this paper.

Precisely speaking, the output constraint $y_2 \geq 0$ in (18) and (19) should be a strict one, i.e., $y_2 > 0$, according to the feasible region conditions. However, this would make the two problems no longer standard MILPs and thus difficult to solve. Here, we relax the strict constraint to a non-strict one to maintain MILP formulations at a slight expense of completeness: the solutions to (18) and (19) are not counterexamples when $y_1 \leq 0$ and $y_2 \geq 0$ simultaneously hold with equality. Fortunately, this problem is minor in practice because the situation where these two equalities simultaneously hold is rare: it can only happen on the boundary of a feasible region. In most cases, such a situation can be avoided by slightly shrinking the feasible region through fine-tuning. Moreover, when solving (18) and (19) with numerical optimizers, such as Gurobi, there are likely to be numerical errors in the inequality constraints, i.e., the constraints are violated by a small amount within a certain tolerance. Therefore, even without relaxation, y_2 has to be strictly less than zero with some margin so that the optimizer can consider the constraint $y_2 > 0$ unsatisfiable and conclude that the problem is infeasible.

In summary, our framework synthesizes a verified neural HJ reachability value function in three stages by solving four subproblems of problem (1), which are defined as follows.

Subproblem 1 (Pre-training). *Train a value network V_θ^π by minimizing loss function* (11).

Subproblem 2 (Counterexample search). *Find constraint counterexamples and invariance counterexamples of a value network V_θ^π by solving problem* (12) *and* (13)*, respectively.*

Subproblem 3 (Fine-tuning). *Fine-tune a value network V_θ^π by minimizing loss function* (16) *on counterexamples found by solving subproblems 2 or 4.*

Subproblem 4 (Verification). *Verify constraint satisfaction condition and forward invariance condition of a value network V_θ^π by solving problem* (18) *and* (19)*, respectively. If both problems are infeasible, return the verified V_θ^π. If either problem is feasible, return the found counterexamples.*

Subproblem 1 is first solved in the pre-training stage. Then, subproblem 2 and 3 are iteratively solved in the adversarial training stage until no counterexample is found from subproblem 2. Next, subproblem 4 is solved in the verification-guided training stage. When counterexamples are found from subproblem 4, they will still be used to solve subproblem 3, and the algorithm returns to the adversarial training stage. This process is repeated until subproblem 4 returns a verified value function.

As mentioned above, we propose three techniques to address the three challenges of improving the scalability of neural value function synthesis and verification. The three techniques correspond to solving subproblems 1, 2, and 3, respectively. The following three subsections provide a detailed introduction to each technique.

4.2 Activation Pattern Alignment

A major difficulty in scaling our proposed framework is the high computational complexity of MILP-based verification. Our experiment shows that verifying a relatively small value network in a four-dimensional system can take more than 2 hours on a common computing platform. To alleviate this problem, we first analyze the reason for such a high computational complexity through the solving mechanism of MILPs. For a neural network with ReLU activation functions, each ReLU unit can be either active or inactive. To handle a neural network constraint, a binary variable is introduced for each ReLU unit to model its activation status, i.e., zero represents an inactive unit, and one represents an active unit, as shown in (20i) and (20j). With these binary variables, MILP problems are solved using a branch-and-bound algorithm. The branching step divides the problem into smaller sub-problems by fixing the values of some binary variables. The bounding step estimates the lower and upper bounds of the objective function in the sub-problems. The computational complexity of the branch-and-bound algorithm is mainly determined by the number of branches to explore, which relies on the number of possible combinations of the binary variable values. Since each binary variable corresponds to a ReLU unit, we can also say that the computational complexity relies on the number of possible activation patterns of the neural network. Each activation pattern corresponds to a linear segment of the neural network. These linear segments divide the input set into different regions, in each of which the neural network is a linear function. Since the number of linear segments largely determines the computational complexity of solving MILPs, we aim to reduce it to accelerate verification.

For a neural network with a given structure, its number of linear segments can vary greatly depending on the network parameter values. For a neural network with N ReLU units, there is at least one linear segment and at most 2^N linear segments in a given input set. To reduce the number of linear segments, we introduce a regularization method called activation pattern alignment (APA) when solving subproblem 1 in pre-training. Suppose we are updating the value network on a batch of states $\{x^{(i)}\}_{i=1}^{N}$. APA first adds a Gaussian noise to each state and obtains a disturbed counterpart of the state:

$$\tilde{x}^{(i)} = x^{(i)} + \xi^{(i)}, \tag{21}$$

where $\xi^{(i)} \sim \mathcal{N}(0, \sigma^2)$. Then, APA computes the following regularization term and adds it to the pre-training loss function (11).

$$L_{\mathrm{APA}}(\theta) = \alpha_{\mathrm{APA}} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{l-1} \sum_{k=1}^{d_j} \frac{\min\{f_k^j(x^{(i)}) \cdot f_k^j(\tilde{x}^{(i)}), 0\}}{\min\{\mathrm{dropgrad}(f_k^j(x^{(i)}) \cdot f_k^j(\tilde{x}^{(i)})), -\epsilon\}}, \tag{22}$$

where $\alpha_{\mathrm{APA}} > 0$ is a coefficient, l is the number of network layers, d_j is the number of neurons in the j-th layer, f_k^j is the value of the k-th neuron in the j-th layer before activation, and ϵ is a small constant for numerical stability. This regularization term encourages states that are close to each other to have similar activation patterns. It takes effect when the activation patterns of $x^{(i)}$ and its disturbed counterpart $\tilde{x}^{(i)}$ are different. In this case, the multiplication of their pre-activation values in the numerator is penalized and driven towards a positive value so that their activation patterns become the same. The last layer of the neural network is excluded when computing L_{APA} because it does not have a ReLU activation function and thus does not influence the number of linear segments. The denominator of L_{APA} does not have a gradient with respect to network parameters θ and only serves as a normalization term.

Reducing the number of linear segments essentially reduces the nonlinearity of a neural network. From this perspective, other neural network regularization methods may also achieve this goal. A widely used regularization method is weight decay, which incorporates an L2 regularization on the network parameters into the optimization process. Another method is the signal-to-noise ratio (SNR) loss proposed by Wei et al. [38], which is designed to reduce the variance and improve the stability of ReLU units to mitigate performance degradation in certified training. Compared with these methods, APA most effectively reduces linear segments while retaining the network approximation ability to the greatest extent, as shown in Section 5.4.1. This is because APA only takes effect when activation patterns of neighboring states are different and does not affect the specific pre-activation values when they have the same sign.

4.3 Boundary-Guided Backtracking

In subproblem 2, we solve two constrained optimization problems (12) and (13) to find counterexamples of the feasible region conditions. Compared with a standard adversarial training problem [17], these two problems not only have a boundary constraint on the optimization variable but also have a non-convex constraint given by the zero-sublevel set of the value network. This zero-sublevel set constraint makes these two problems difficult to solve because most existing adversarial training methods, such as the fast gradient sign method [17] and projected gradient descent (PGD) method [27], cannot directly handle such non-convex constraints.

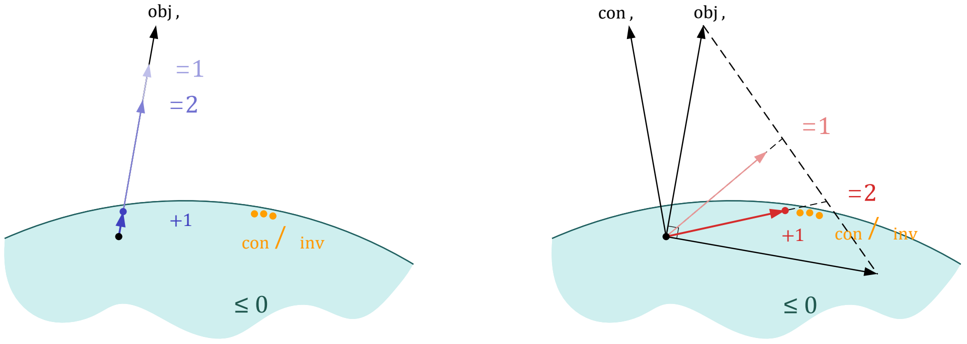
A straightforward method for handling the zero-sublevel set constraint is to perform a backtracking line search in each PGD iteration, resulting in PGD with backtracking (PGD-B). Specifically, we start the search from an initial state $x_{(0)}$ randomly sampled in the zero-sublevel set. In each iteration, we perform a backtracking line search along the gradient of the objective function, followed by a projection operation until the resulting state is in the zero-sublevel set:

$$x_{(k+1)} = \Pi_{\mathcal{X}}(x_{(k)} + \eta^s g_{\text{obj},(k)}), \tag{23}$$

where $\Pi_{\mathcal{X}}$ is the projection operator on $\mathcal{X}$, $\eta \in (0, 1)$ is a constant, $s \in \mathbb{N}$ is the smallest number such that $V_\theta^\pi(x_{(k+1)}) \leq 0$, and $g_{\text{obj},(k)}$ is the unit vector of the gradient of the objective function. Take problem (12) as an example,

$$g_{\text{obj},(k)} = \frac{\nabla_x h(x)}{\|\nabla_x h(x)\|_2}\Big|_{x=x_{(k)}}. \tag{24}$$

The problem with PGD-B is that the search becomes very inefficient when approaching the boundary of the zero-sublevel set. This is because the gradient of the objective function becomes almost vertical to the boundary, i.e., in the same direction as the gradient of the constraint function. This is obvious for problem (13) because the gradients of $V_\theta^\pi(f_\pi(x))$ and $V_\theta^\pi(x)$ are very similar as long as the time step of the system is not very large. For problem (12), this phenomenon occurs when the boundary of the zero-sublevel set overlaps or is very close to that of the constrained set. When the gradient becomes vertical to the boundary, the backtracking line search will end up in very small step sizes or even stop to avoid stepping out of the zero-sublevel set, as shown in Figure 2(a). It seems to be a minor problem since the counterexamples we are searching for are also located near the boundary of the zero-sublevel set; otherwise, the state cannot leave the zero-sublevel set or violate the constraint in one step unless it is already very close to the boundary. However, practically, PGD-B would get stuck somewhere not exactly at a counterexample because the initial state is randomly chosen and counterexamples are sparsely distributed. In this case, even if close to a counterexample, PGD-B may never find it because PGD-B can only search towards the boundary but not along it.



(a) PGD-B

(b) BGB

Fig. 2. Search steps of PGD-B and BGB. The black dot represents the current position of the search, and the orange dots represent possible counterexamples. (a) PGD-B always searches along the gradient of the objective function and gets stuck near the boundary of the feasible region. (b) BGB rotates the search direction towards the tangent plane of the boundary and effectively search along the boundary, thus finding counterexamples more efficiently.

To solve the problem of PGD-B, we propose a boundary-guided backtracking (BGB) method that can efficiently search counterexamples along the boundary. This is achieved by rotating the line search direction towards the tangent plane of the boundary when approaching it. Specifically, instead of performing the line search along $g_{\text{obj},(k)}$, BGB computes the search direction as a weighted sum of $g_{\text{obj},(k)}$ and another unit vector $b_{(k)}$, as shown in Figure 2(b). $b_{(k)}$ is perpendicular to $g_{\text{obj},(k)}$, coplanar with both $g_{\text{obj},(k)}$ and $g_{\text{con},(k)}$ and makes an obtuse angle with $g_{\text{con},(k)}$, where $g_{\text{con},(k)}$ is the unit vector of the gradient of the constraint function, i.e.,

$$g_{\text{con},(k)} = \frac{\nabla_x V_\theta^\pi(x)}{\|\nabla_x V_\theta^\pi(x)\|_2}\bigg|_{x=x_{(k)}}. \tag{25}$$

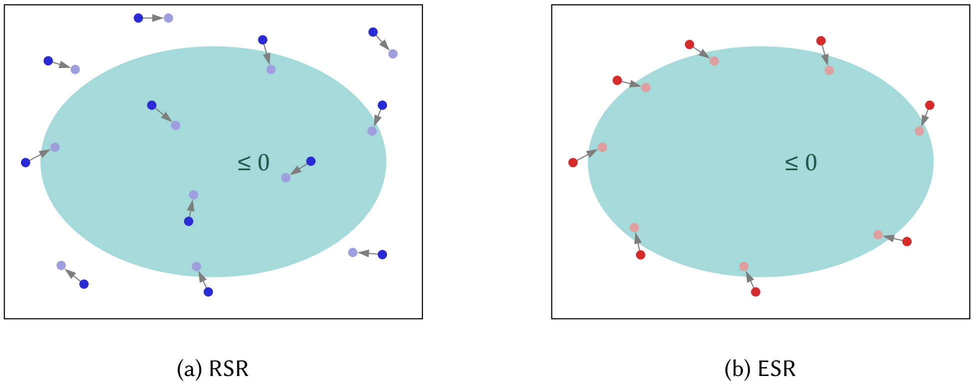
Then, $b_{(k)}$ is computed as

$$b_{(k)} = \frac{\hat{b}_{(k)}}{\|\hat{b}_{(k)}\|_2}, \quad \hat{b}_{(k)} = (g_{\text{obj},(k)} \cdot g_{\text{con},(k)})g_{\text{obj},(k)} - g_{\text{con},(k)}. \tag{26}$$

BGB line search is performed as

$$x_{(k+1)} = \Pi_{\mathcal{X}}(x_{(k)} + \eta_l^s(\eta_a^s g_{\text{obj},(k)} + (1-\eta_a^s)b_{(k)})), \tag{27}$$

where $\eta_l, \eta_a \in (0,1)$ are backtracking discounts for step size and search direction, respectively. The key of BGB is the unit vector $b_{(k)}$, which determines the changing range of search direction. We set $b_{(k)}$ vertical to $g_{\text{obj},(k)}$ to ensure that the search direction always makes a sharp angle with $g_{\text{obj},(k)}$ so that the objective function is always increased. The reason for setting $b_{(k)}$ coplanar with both $g_{\text{obj},(k)}$ and $g_{\text{con},(k)}$ is that this is the quickest way to rotate the search direction from $g_{\text{obj},(k)}$ to the tangent plane of the boundary. Vector $b_{(k)}$ should make an obtuse angle with $g_{\text{con},(k)}$ because this is the direction where the constraint function decreases. Compared with a standard backtracking line search, BGB not only decreases the step size but also rotates the search direction towards the boundary in each iteration. Altering the search direction away from the gradient may seem counterintuitive since it slows the convergence of objective function in an unconstrained case. However, it turns out to be more efficient when constraint exists because of a significant reduction in the number of backtracking steps. While



(a) RSR

(b) ESR

Fig. 3. Regularized states of RSR and ESR. (a) The darker blue dots represent regularized states and the lighter ones are their next states. The regularized states are randomly sampled in the state space and may be infeasible. (b) The darker red dots represent regularized states and the lighter ones are their next states. The regularized states are entering states and must be feasible.

the theoretical convergence speed of BGB requires further analysis, our experiments show that in practice, it effectively avoids getting stuck near the boundary and enables efficient search along the boundary. This ability greatly improves the efficiency of finding counterexamples and accelerates adversarial training, as evident in Section 5.4.2.

4.4 Entering State Regularization

We discover in our experiments that when fine-tuning the value network in subproblem 3, directly minimizing loss functions (14) and (15) on counterexamples will result in severe shrinkage of the zero-sublevel set. Similar phenomena are also observed in other works on safety certificate synthesis [10, 26]. These works deal with this problem by adding a regularization term that minimizes the output of the value network on randomly sampled states, as shown in Figure 3(a). This method is called random state regularization (RSR), and the mathematical formula of its regularization term is

$$L_{\text{RSR}}(\theta) = \alpha_{\text{RSR}} \frac{1}{N_{\text{rnd}}} \sum_{i=1}^{N_{\text{rnd}}} V_\theta^\pi(x_{\text{rnd}}^{(i)}), \tag{28}$$

where α_{RSR} is a coefficient, $x_{\text{rnd}}^{(i)}$ is the i-th state uniformly sampled from the state space, and N_{rnd} is the number of sampled states. The problem with RSR is that it may cause some infeasible states to be mistakenly included in the zero-sublevel set, resulting in violations of the feasible region conditions. To avoid this problem, we only regularize states that do not compromise the satisfaction of feasible region conditions when included in the zero-sublevel set. According to the definition of a feasible region, we can derive the following theorem, which provides a method for expanding a feasible region.

Theorem 4 (Feasible region expansion). *Let* X^π *be a feasible region of* π. $\forall x \in \mathcal{X} \setminus \mathrm{X}^\pi$, *if* $h(x) \le 0$ *and* $f_\pi(x) \in \mathrm{X}^\pi$, $\mathrm{X}^\pi \cup \{x\}$ *is also a feasible region of* π.

Theorem 4 tells us that if a state outside the feasible region satisfies the state constraint and enters the feasible region in one step, it can be included in the feasible region. Such states are called *entering states*. The core idea of our regularization method, called entering state regularization (ESR), is to include entering states into the zero-sublevel set of the value network, as shown in Figure 3(b). In model predictive control, there is a concept similar to entering state called precursor set, which is defined as the set of all states whose next state is in the current set [7]. The precursor set is used for computing the maximal control invariant set: start from the whole state space as the initial set and iteratively intersect the current set with its precursor set, the resulting set gradually shrinks and converges to the maximal control invariant set. Our entering states are those in the precursor set but not in the current set, and they are used for enlarging a feasible region instead of shrinking it. To perform ESR, we first randomly sample some states in the state space in each iteration and then filter out entering states that satisfy:

$$h(x) \leq -\delta,\ V_\theta^\pi(x) > 0,\ V_\theta^\pi(f_\pi(x)) \leq -\delta, \tag{29}$$

where δ is a small positive constant. The purpose of introducing this constant is to avoid the undesirable influence of the regularization on nearby states. Specifically, due to the generalization ability of neural networks, when including an entering state into the zero-sublevel set, some of its nearby states may also be included. These nearby states may not be entering states and may cause violation of the feasible region conditions. By introducing δ, we set a margin of entering states to the boundary of the constrained set and the zero-sublevel set, thus decreasing the probability of mistaken inclusion. Using the filtered entering states, we compute the following regularization term and add it to the value loss function.

$$L_{\text{ESR}}(\theta) = \alpha_{\text{ESR}} \frac{1}{N_{\text{ent}}} \sum_{i=1}^{N_{\text{ent}}} V_\theta^\pi(x_{\text{ent}}^{(i)}), \tag{30}$$

where α_{ESR} is a coefficient, N_{ent} is the number of entering states, and $x_{\text{ent}}^{(i)}$ stands for the i-th entering state. Minimizing (30) will decrease the values of entering states until they become negative, in which case they are included in the zero-sublevel set and will no longer be identified as entering states. It is worth mentioning that the states filtered out by (29) are not necessarily feasible because the value network has not been verified yet. For example, it is possible that the next state $f_\pi(x)$, which is currently in the zero-sublevel set, is excluded from the set in later iterations, making the current state x also infeasible. Therefore, this regularization method may also cause mistaken inclusion of infeasible states. However, our method is based on the fact that the value network is pre-trained, which ensures that the zero-sublevel set does not deviate much from the feasible region. This greatly decreases the probability of including infeasible states. Moreover, since the zero-sublevel set usually shrinks during fine-tuning and soon becomes smaller than the feasible region, the filtered entering states are feasible in most cases. Compared with ESR, existing regularization methods are more harmful to the feasible region conditions because they use randomly sampled states for regularization, which are more likely to include infeasible states.

4.5 Analysis of Proposed Framework

In this subsection, we analyze some important properties and assumptions of our proposed value function synthesis framework. First, we study the soundness and completeness of the verification in our framework. In our problem, soundness means that when MILP (18) is infeasible, the constraint satisfaction condition (7) actually holds, and when MILP (19) is infeasible, the forward invariance condition (8) actually holds. Completeness means that when either of the two problems is feasible, its corresponding condition is actually violated. According to the derivation of MILPs (18) and (19), except for the strictness of their output constraints, their infeasibilities are equivalent to the satisfaction of conditions (7) and (8), respectively. Therefore, our method is sound and complete, but its implementation is subject to floating point error induced incompleteness, as discussed in Section 4.1.

Second, we discuss the assumptions made by our framework. We assume the availability of an accurate dynamic model. Although the model can be represented by a neural network, we do not consider its approximation error. However, it is worth mentioning that this paper focuses on establishing the first method to obtain verifiable neural value functions for HJ reachability analysis. We choose to use the perfect model to study the synthesis and verification approach. The model assumptions could be relaxed to account for uncertainties, disturbances, and unmodeled dynamics. Specifically, we can introduce an additive term with bounded norm to the known dynamic model:

$$\tilde{f}(x,u) = f(x,u) + \Delta f, \quad \|\Delta f\| \leq \zeta, \tag{31}$$

where $\tilde{f}$ is the true dynamic model, f is the known part, and Δf is the unmodeled part with norm bounded by $\zeta \in \mathbb{R}^{+}$. When formulating the forward invariance verification problem, Δf can be encoded as a variable optimized together with the state to find counterexamples:

$$\begin{aligned} \text{find} \quad & x, \Delta f && \text{(32a)} \\ \text{s.t.} \quad & x \in \mathcal{X}, \|\Delta f\| \leq \zeta, && \text{(32b)} \\ & y = \mathrm{M}_{\mathrm{inv}}(x) = \begin{bmatrix} V_\theta^\pi(x) \\ V_\theta^\pi(f_\pi(x) + \Delta f) \end{bmatrix}, && \text{(32c)} \\ & y_1 \leq 0, y_2 \geq 0. && \text{(32d)} \end{aligned}$$

To keep the problem still an MILP, we can choose the L1 norm or infinity norm, which can be expressed by linear equalities. If this problem is infeasible, it is guaranteed that the forward invariance condition is satisfied under any possible uncertainty. This robust version problem formulation can also be used for verifying systems with stochastic dynamics. However, detailed studies, e.g., how good the proposed framework will perform in these situations with non-perfect models, will be left for future work. Besides a known deterministic model, our framework also requires a fixed control policy, i.e., we only verify the feasible region under a fixed policy. For the optimal policy, the verification of the forward invariance condition becomes a max-min problem where the inner minimization considers the best-case control input. The minimization can be approximated by a Taylor model with bounded remainders as proposed by Hu et al. [21]. For time-varying or adaptive controllers, verifying a fixed forward invariance condition is no longer sufficient since the state transition dynamics are changing. A possible solution is to bound the controller's output on each state and verify the condition under all possible outputs. In addition, we only consider the ReLU activation function in this work because it is piecewise linear, allowing us to formulate the verification problem as an MILP. While other smooth activation functions or architectures might improve network approximation ability, they introduce nonlinearities that break the MILP formulation. For such cases, other verification methods, e.g., α, β-CROWN, can also be employed and integrated into our framework, although at the cost of increased computational overhead or loss of verification completeness.

Third, we study the relationship between the zero-sublevel set of a verified V_θ^π, denoted as $\mathrm{X}_{V_\theta^\pi}$, and the feasible region of π. Since a verified V_θ^π satisfies conditions (7) and (8), which are necessary and sufficient conditions for a feasible region, $\mathrm{X}_{V_\theta^\pi}$ is a feasible region of π. Since the feasible region satisfying conditions (7) and (8) is not unique, $\mathrm{X}_{V_\theta^\pi}$ may not be the maximum feasible region of π but only an under-approximation of it. Nevertheless, being a feasible region of π, $\mathrm{X}_{V_\theta^\pi}$ already guarantees that trajectories starting from inside it and sampled under π are safe in the long term. This enables us to use V_θ^π and π to construct a safety filter in which V_θ^π is a safety monitor and π is a backup policy [19]. Specifically, starting from inside $\mathrm{X}_{V_\theta^\pi}$, the safety filter checks in each step whether the next state after applying some nominal action is still in $\mathrm{X}_{V_\theta^\pi}$. If this is true, the nominal action is applied. Otherwise, the nominal action is replaced by the action computed by π, which ensures by forward invariance condition that the next state is still in $\mathrm{X}_{V_\theta^\pi}$. This safety filter can ensure strict long-term constraint satisfaction for an arbitrary policy.

Last, we discuss the stability of our training loop, i.e., whether it is guaranteed to find a verified value function and under what conditions it will fail. Although our framework successfully synthesizes verified value functions on various tasks in our experiments (See Section 5), such success is not guaranteed. In some cases, the training loop may not be able to find a verified value function with a non-trivial zero-sublevel set, even if such a true value function exists. The reasons for the failure of synthesis are multiple, and we list four main potential failure modes as follows.

(1) **Instability of adversarial training.** There are always counterexamples found in adversarial training, and they cannot be eliminated. As a result, verification cannot start before the maximum number of iterations is exceeded. This may be because the counterexample search is too inefficient, or catastrophic forgetting happens in adversarial training.
(2) **Instability of verification-guided training.** No counterexamples can be found in adversarial training, but verification always fails. This indicates that counterexamples exist and the search algorithm is not effective enough to find them.
(3) **Inefficient verification.** Verification takes so long that the time limit is exceeded. This is because the computational complexity of solving the MILPs is too high.
(4) **Converging to invalid local optima.** The value function becomes all positive in the state space, and its zero-sublevel set becomes empty. This is because too many feasible states are mistakenly excluded from the zero-sublevel set during fine-tuning, causing the set to shrink so much that it disappears.

In essence, these failure modes all stem from the three challenges to improve scalability mentioned in Section 1. While our proposed three techniques address these challenges to some extent, they persist as the problem's dimension increases.

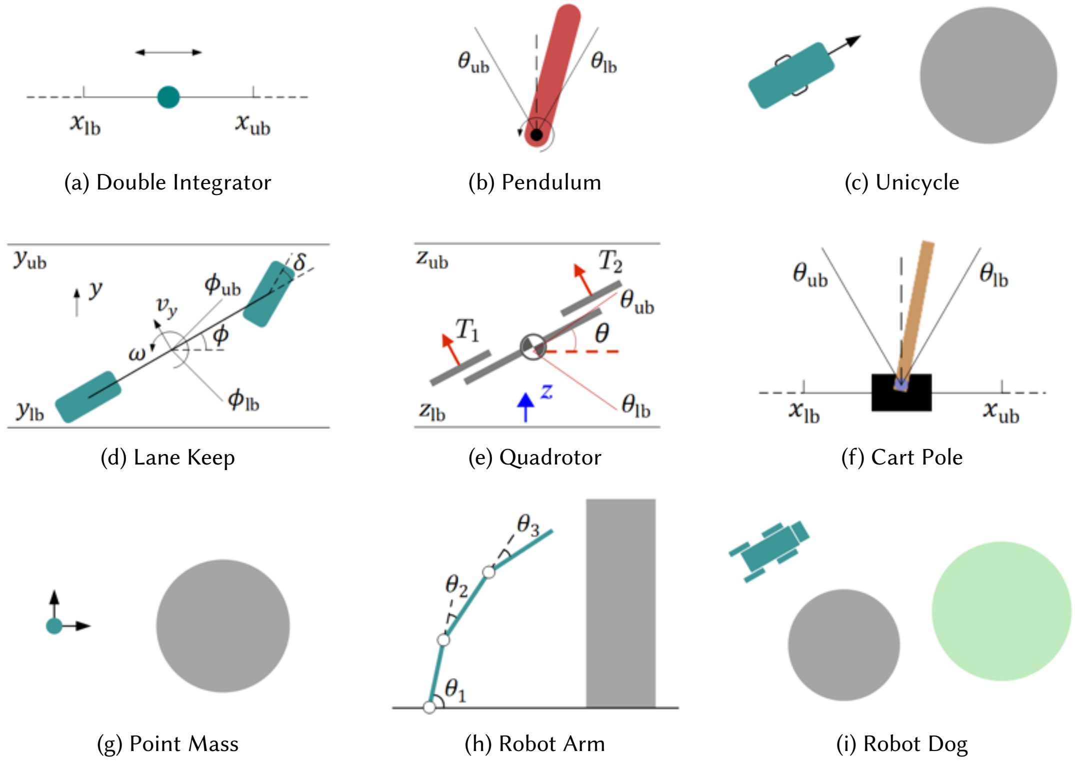
5 Experiments

Through experiments, we aim to answer the following questions: 1) Can our proposed framework synthesize verified neural HJ reachability value functions on different safe control tasks? 2) Can APA accelerate verification, and how does it perform compared with other neural network regularization methods? 3) Can BGB accelerate counterexample search, and how does it perform compared with other search methods? 4) Can ESR enlarge feasible region, and how does it perform compared with other feasible region regularization methods? We answer the first question by testing our framework on nine commonly used safe control tasks. We answer the remaining three questions by comparing our proposed three techniques with several existing methods. Before that, we first introduce a neural safety certificate synthesis and verification benchmark and some implementation details of our framework.

5.1 Cersyve-9 Benchmark

There have been several benchmarks for neural network verification [8], including image classification datasets such as MNIST [24] and CIFAR [23], vehicle collision prediction problem [15], and aircraft collision avoidance system ACAS Xu [31]. However, these benchmarks are incompatible with the verification of neural safety certificates because of the underlying differences between their problems and safe control tasks. Specifically, safe control tasks require neural networks to satisfy certain properties everywhere in the state space, i.e., the input set of the verification problem is the entire state space, while existing benchmarks only consider verification in either a small disturbance set around data samples [24, 23, 15] or part of the state space [31]. In addition, verifying neural safety certificates involves system dynamics and control policies, which requires certain conversions like (18) and (19) before it can be formulated as standard verification problems while existing benchmarks only consider verification of a single neural network.

To bridge this gap, we provide a benchmark called Cersyve-9 for neural safety Certificate synthesis and verification in safe control tasks. Cersyve-9 contains: (1) Nine commonly used control tasks with state dimensions ranging from two to six, as shown in Figure 4 and Table 1. These tasks include both linear and nonlinear dynamics and safety constraints. (2) A set of neural safety certificate synthesis tools, including pre-training, adversarial



(a) Double Integrator (b) Pendulum (c) Unicycle

(d) Lane Keep (e) Quadrotor (f) Cart Pole

(g) Point Mass (h) Robot Arm (i) Robot Dog

Fig. 4. Snapshots of safe control tasks in Cersyve-9. The grey objects in Unicycle, Point Mass, Robot Arm, and Robot Dog are obstacles. The green circle in Robot Dog is the goal.

training, and verification-guided training modules, as well as evaluation tools for synthesized certificates. These tools facilitate secondary development and performance comparison of different synthesis algorithms. (3) An MILP-based neural safety certificate verification algorithm, as well as neural value functions synthesized and verified by our framework on all nine tasks for comparing different verification algorithms.

5.1.1 Task Descriptions. Each task in Cersyve-9 defines state space, control input space, dynamic model, and safety constraints, which are detailed as follows.

Double Integrator requires stabilizing a second-order linear system to the origin under boundary constraints on the position. The state of this task is $x = [p, \dot{p}]^\top \in \mathbb{R}^2$, where p is the position. The control input is $u \in \mathbb{R}$. The state space and control input space are hyperrectangles specified by $[x_{\min}, x_{\max}]$ and $[u_{\min}, u_{\max}]$, respectively. All following tasks adopt the setting of hyperrectangular state and control input spaces. The dynamic model of this task is

$$\begin{aligned} p_{t+1} &= p_t + \dot{p}_t \Delta t, \\ \dot{p}_{t+1} &= \dot{p}_t + u_t \Delta t, \end{aligned} \tag{33}$$

where Δt is the time step. The safety constraint is $p_{\text{lb}} \leq p \leq p_{\text{ub}}$, where p_{lb} and p_{ub} are lower and upper bounds on the position, respectively.

Pendulum requires stabilizing a pendulum to the upright position under boundary constraints on its angle [9]. The state of this task is $x = [\theta, \dot{\theta}]^\top \in \mathbb{R}^2$, where θ is the angle of the pendulum. The control input $u \in \mathbb{R}$ is the

Table 1. Information of safe control tasks in Cersyve-9. In Robot Dog, we directly fit the closed-loop dynamic model with a neural network, and thus, the control dimension is irrelevant to value network synthesis and verification.

Task	State dim	Control dim	Dynamics	Constraint
Double Integrator	2	1	Linear	Linear
Pendulum	2	1	Nonlinear	Linear
Unicycle	3	2	Nonlinear	Nonlinear
Lane Keep	4	1	Linear	Linear
Quadrotor	4	2	Nonlinear	Linear
Cart Pole	4	1	Nonlinear	Linear
Point Mass	4	2	Nonlinear	Nonlinear
Robot Arm	6	3	Linear	Nonlinear
Robot Dog	5	/	Nonlinear	Nonlinear

torque applied to the pendulum. The dynamic model is

$$\begin{aligned}\theta_{t+1} &= \theta_t + \dot{\theta}_{t+1}\Delta t,\\ \dot{\theta}_{t+1} &= \dot{\theta}_t + \left(\frac{3g}{2l}\sin\theta_t + \frac{9u}{ml^2}\right)\Delta t,\end{aligned} \tag{34}$$

where m is the mass of the pendulum, l is its length, and g is the gravitational acceleration. The safety constraint is $\theta_{\mathrm{lb}} \le \theta \le \theta_{\mathrm{ub}}$.

Unicycle requires controlling a unicycle model to avoid collision with a circular obstacle. The state of this task is $x = [v, x_{\mathrm{o}}, y_{\mathrm{o}}]^\top \in \mathbb{R}^3$, where v is the velocity angle of the unicycle, and $x_{\mathrm{o}}, y_{\mathrm{o}}$ is the position of the obstacle in the unicycle frame. The control input is $u = [a, \omega]^\top \in \mathbb{R}^2$, where a is the acceleration of the unicycle, and ω is its angular velocity. The dynamic model is

$$\begin{aligned}v_{t+1} &= v_t + a_t\Delta t,\\ x_{\mathrm{o}(t+1)} &= (x_{\mathrm{o}t} - v_t\Delta t)\cos(\omega_t\Delta t) + y_{\mathrm{o}t}\sin(\omega_t\Delta t),\\ y_{\mathrm{o}(t+1)} &= y_{\mathrm{o}t}\cos(\omega_t\Delta t) - (x_{\mathrm{o}t} - v_t\Delta t)\sin(\omega_t\Delta t).\end{aligned} \tag{35}$$

The safety constraint is $\sqrt{x_{\mathrm{o}}^2 + y_{\mathrm{o}}^2} \ge r_{\mathrm{o}}$, where r_{o} is the radius of the obstacle.

Lane Keep requires keeping a 2DOF vehicle dynamics model in a straight line under boundary constraints on its lateral position and heading angle. The state of this task is $x = [y, \phi, v_y, \omega]^\top \in \mathbb{R}^4$, where y is the lateral position of the vehicle, ϕ is the heading angle, v_y is the lateral velocity, and ω is the angular velocity. The control input $u = \delta \in \mathbb{R}$ is the front wheel angle. The dynamic model is

$$\begin{aligned}y_{t+1} &= y_t + (\phi_t v_x + v_y)\Delta t,\\ \phi_{t+1} &= \phi_t + \omega_t\Delta t,\\ v_{y(t+1)} &= \left(1 + \frac{k_1 + k_2}{mv_x}\Delta t\right)v_{yt} + \left(\frac{ak_1 - bk_2}{mv_x} - v_x\right)\omega_t\Delta t,\\ \omega_{t+1} &= \frac{ak_1 - bk_2}{I_z v_x}v_{yt}\Delta t + \left(1 + \frac{k_1a^2 + k_2b^2}{I_z v_x}\Delta t\right)\omega_t,\end{aligned} \tag{36}$$

where k_1 and k_2 are the cornering stiffness of the front and rear wheels, respectively, a and b are the distance from the center of gravity to the front and rear axles, respectively, m is the mass of the vehicle, I_z is the moment of inertia on the vertical axis, and v_x is the longitudinal velocity, which is a constant value. The safety constraints are $y_{\text{lb}} \leq y \leq y_{\text{ub}}$ and $\phi_{\text{lb}} \leq \phi \leq \phi_{\text{ub}}$.

Quadrotor requires controlling a 2D quadrotor to a hover position under boundary constraints on its vertical position and roll angle. The state of this task is $x = [z, \theta, \dot{z}, \dot{\theta}]^\top \in \mathbb{R}^4$, where z is the vertical position of the quadrotor, and θ is the roll angle. The control input $u = [T_1, T_2]^\top \in \mathbb{R}^2$ represents the thrust forces exerted by the rotors. The dynamic model is

$$\begin{aligned} z_{t+1} &= z_t + \dot{z}_t \Delta t, \\ \theta_{t+1} &= \theta_t + \dot{\theta}_t \Delta t, \\ \dot{z}_{t+1} &= \dot{z}_t + \left(\frac{(T_{1t} + T_{2t}) \cos \theta_t}{m} - g \right) \Delta t, \\ \dot{\theta}_{t+1} &= \dot{\theta}_t + \frac{(T_{2t} - T_{1t}) d}{I_y} \Delta t, \end{aligned} \tag{37}$$

where m is the mass of the quadrotor, d is the diameter, and I_y is the moment of inertia. The safety constraints are $z_{\text{lb}} \leq z \leq z_{\text{ub}}$ and $\theta_{\text{lb}} \leq \theta \leq \theta_{\text{ub}}$.

Cart Pole requires balancing a pole on a moving cart to the upright position under boundary constraints on the cart position and the pole angle [9]. The state of this task is $x = [y, \theta, \dot{y}, \dot{\theta}]^\top \in \mathbb{R}^4$, where y is the cart position, and θ is the pole angle. The control input $u = F \in \mathbb{R}$ is the force applied to the cart. The dynamic model is

$$\begin{aligned} y_{t+1} &= y_t + \dot{y}_t \Delta t, \\ \theta_{t+1} &= \theta_t + \dot{\theta}_t \Delta t, \\ \dot{y}_{t+1} &= \dot{y}_t + \frac{F + ml\dot{\theta}_t^2 \sin \theta_t - ml\ddot{\theta}_t \cos \theta_t}{M} \Delta t, \\ \dot{\theta}_{t+1} &= \dot{\theta}_t + \ddot{\theta}_t \Delta t, \end{aligned} \tag{38}$$

where m is the mass of the pole, M is the total mass of the pole and the cart, l is the length of the pole, and

$$\ddot{\theta}_t = \frac{3Mg \sin \theta_t - 3(F + ml\dot{\theta}_t^2 \sin \theta_t) \cos \theta_t}{(4M - 3m \cos^2 \theta_t) l}.$$

The safety constraints are $y_{\text{lb}} \leq y \leq y_{\text{ub}}$ and $\theta_{\text{lb}} \leq \theta \leq \theta_{\text{ub}}$.

Point Mass has the same setting as Unicycle except that the model is changed to a 2D point mass. The state of this task is $x = [v_x, v_y, x_{\text{o}}, y_{\text{o}}]^\top \in \mathbb{R}^4$, where v_x and v_y are the velocities on the x and y axes, respectively. The control input is $u = [a, \omega]^\top \in \mathbb{R}^2$. The dynamic model is

$$\begin{aligned} v_{x(t+1)} &= (v_{xt} + a_t \Delta t) \cos(\omega_t \Delta t) + v_{yt} \sin(\omega_t \Delta t), \\ v_{y(t+1)} &= v_{yt} \cos(\omega_t \Delta t) - (v_{xt} + a_t \Delta t) \sin(\omega_t \Delta t), \\ x_{\text{o}(t+1)} &= (x_{\text{o}t} - v_t \Delta t) \cos(\omega_t \Delta t) + y_{\text{o}t} \sin(\omega_t \Delta t), \\ y_{\text{o}(t+1)} &= y_{\text{o}t} \cos(\omega_t \Delta t) - (x_{\text{o}t} - v_t \Delta t) \sin(\omega_t \Delta t). \end{aligned} \tag{39}$$

The safety constraint is $\sqrt{x_{\text{o}}^2 + y_{\text{o}}^2} \geq r_{\text{o}}$.

Robot Arm requires controlling a robot arm with three joints to a target position while avoiding collision with a wall in the front. The state of this task is $x = [\theta_1, \theta_2, \theta_3, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3]^\top \in \mathbb{R}^6$, where θ_1 is the angle of the first joint, and θ_2 and θ_3 are the incremental angles of the second and third joints relative to their previous joints. The control input $u = [\ddot{\theta}_1, \ddot{\theta}_2, \ddot{\theta}_3]^\top \in \mathbb{R}^3$ represents the angular accelerations of the three joints. The dynamic model is obtained through forward Euler discretization and is omitted here. The safety constraint is that the length of the robot arm's projection on the horizontal axis must not exceed a specific threshold:

$$\sum_{i=1}^{3} \cos\left(\sum_{j=1}^{i} \theta_j\right) \leq 1.5. \tag{40}$$

Robot Dog is a robot locomotion task designed by He et al. [18], which requires controlling a robot dog to reach a goal while avoiding obstacles on its way. The state of this task is $x = [v, x_{\text{g}}, y_{\text{g}}, x_{\text{o}}, y_{\text{o}}]^\top \in \mathbb{R}^5$, where v is the velocity of the robot dog, $x_{\text{g}}, y_{\text{g}}$ is the position of the goal, and $x_{\text{o}}, y_{\text{o}}$ is the position of the obstacle. In this task, we directly fit a closed-loop dynamic model with a neural network on data collected by an RL policy, and thus, the control input is irrelevant to value function synthesis and verification. The safety constraint is $\sqrt{x_{\text{o}}^2 + y_{\text{o}}^2} \geq r_{\text{o}}$. He et al. [18] propose a method called Agile But Safe (ABS) that learns a neural HJ reachability value function and a safe policy without verification. While our experiments are only performed in simulation, ABS itself is evaluated in the real world.

5.1.2 Implementation Details. For tasks with linear dynamic models and constraint functions, we directly use their analytical forms for value network synthesis. For tasks with nonlinear dynamics or constraints, we fit the nonlinear dynamics or constraints with neural networks for synthesis. We design linear control policies with control limits for all tasks except Robot Dog, where we use the neural network policy trained by ABS. In Robot Dog, we directly fit the closed-loop dynamic model with a neural network, and thus, the control dimension is irrelevant to value network synthesis. In other tasks with nonlinear dynamics, we fit the open-loop dynamic models with neural networks and substitute linear control policies to obtain closed-loop dynamics. In this way, all functions involved in verification are piecewise linear, so the verification problems can be formulated as MILPs. In theory, we could use the original nonlinear versions of the dynamics and constraints by calling some verifiers for nonlinear cyber-physical systems [22, 35]. The reason why we approximate these functions with piecewise linear neural networks is to better integrate with existing verification tools. Our neural network approximation of dynamics and constraints did introduce model mismatch to real-world robot dynamics, but that does not affect the validity of our benchmark because we can think of synthesis and verification as inherently performed on systems with approximated dynamics and constraints. It will be our future work to investigate exact synthesis and verification with respect to nonlinear dynamics and constraints.

We follow the practice of Nagabandi et al. [30] to train neural network dynamic models. We use a neural network $f_\phi(x, u)$ to parameterize the change of state in a time step, i.e., the predicted next state is $\hat{x}_{t+1} = x_t + f_\phi(x_t, u_t)$. Training data is collected by uniformly sampling initial states in the state space and executing random control inputs at every time step. The collected data is recorded in the form of state transition pairs, i.e., $\mathcal{D} = \{(x^{(i)}, u^{(i)}, x'^{(i)})\}_{i=1}^N$, where N is the number of data. To ensure the loss function weights different state elements equally, we subtract the mean and divide by the standard deviation of the data. We then add zero-mean Gaussian noise with a standard deviation of 0.01 to all data. We train the dynamic model by minimizing the following loss function.

$$L_{\text{dyn}}(\phi) = \frac{1}{N} \sum_{i=1}^{N} \|(x'^{(i)} - x^{(i)}) - f_\phi(x^{(i)}, u^{(i)})\|_2^2. \tag{41}$$

For Robot Dog, we directly train a closed-loop dynamic model $f_\phi(x)$ with training data collected by a neural network policy. Data preprocessing and loss function of Robot Dog are similar to those of other tasks.

We use the same neural network structure in all tasks. Both the dynamics and value networks have two hidden layers with 32 neurons each. The constraint network has two hidden layers with 16 neurons each. All experiments are performed on an AMD Ryzen 7 5800 8-Core CPU. Other hyperparameters are listed in Table 2. The hyperparameters introduced by our algorithm include those related to APA, BGB, and ESR. The APA coefficients are chosen to balance the neural network's linearity and performance, with its sensitivity analysis

Table 2. Detailed hyperparameters.

Stage	Hyperparameter	Value
Pre-training	Learning rate for dynamics network	1e-3
	Learning rate for value network	3e-4
	Batch size	256
	Training epochs for dynamics network	100
	Iterations for value network	100000
	Discount factor for value network	0.9
	Weight decay	1e-3
	APA coefficient for dynamics network	0.01
	APA coefficient for value network	1e-4
	APA constant ϵ	1e-4
	SNR coefficients for dynamics network	(0, 1e-3)
	SNR coefficients for value network	(0, 0.1)
	APA & SNR noise scale	0.1
Adversarial training	Learning rate	1e-4
	Max iteration	100000
	Batch size for counterexample search	1000
	PGD steps per iteration	10
	PGD step size	0.1
	Backtracking steps	20
	BGB search direction discount	0.5
	BGB step length discount	0.8
	ESR coefficient	0.1
	ESR sample batch size	1000
	ESR margin δ	0.01

performed in Section 5.4.1. The APA noise scale also controls the regularization strength—a larger noise scale results in stronger regularization. Analysis of how the noise scale influences performance can be found in the SNR paper [38]. The BGB search direction and step length discounts affect the counterexample search efficiency. Larger discounts result in finer but slower backtracking, while smaller discounts result in faster and sparser backtracking. The choice of the step length discount follows that in standard backtracking algorithms. For the search direction discount, we find that a relatively small value is more efficient because the direction of counterexamples usually deviates a lot from the gradient near the boundary. The ESR coefficient balances the efficiency of counterexample elimination and the severity of feasible region shrinkage. However, since we only regularize entering states and avoid counterexamples, a relatively large coefficient works well in practice. The ESR batch size simply follows the batch size of adversarial training. The ESR margin is set to avoid false identification of entering states. A larger margin is safer but reduces the regularization strength. Its value is chosen to balance region shrinkage and training efficiency.

5.2 Evaluation Procedure and Metrics

To evaluate synthesis results, we consider three aspects: 1) counterexample search efficiency, 2) verification efficiency, and 3) size of feasible region.

Table 3. Neural HJ reachability value function synthesis results of our proposed framework.

Task	FT iter (k)	FT time (s)	# Verify	Verify time (s)	TFR
Double Integrator	1.2	28.5	1	0.5	0.940
Pendulum	1.1	31.4	1	3.9	0.962
Unicycle	6.9	175.0	1	11.6	0.911
Lane Keep	5.9	184.7	4	1.4	0.750
Quadrotor	1.6	55.3	1	2.9	0.906
Cart Pole	2.4	83.3	1	171.5	0.404
Point Mass	8.6	321.7	4	48.2	0.594
Robot Arm	25.2	853.8	15	5.5	0.403
Robot Dog	6.3	333.7	2	311.9	0.872

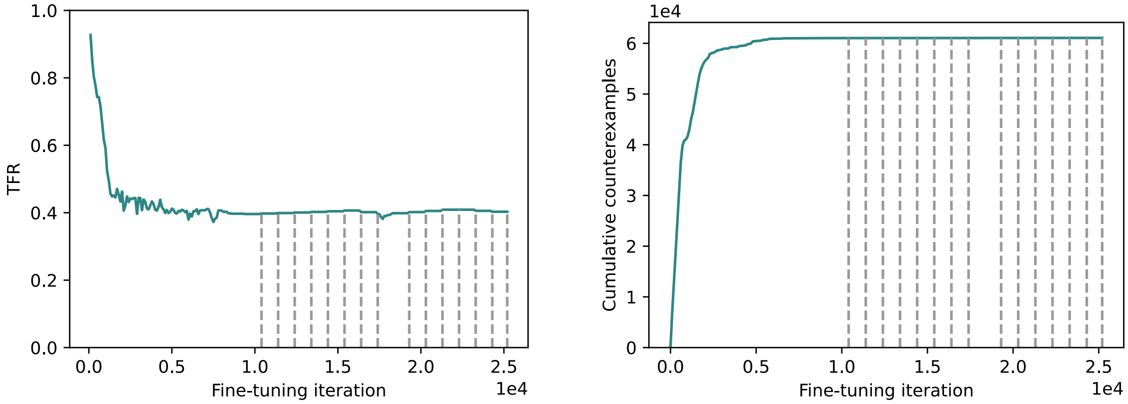
For counterexample search efficiency, we count the number of fine-tuning iterations, total fine-tuning time, and the number of verifications. A smaller number of fine-tuning iterations means more counterexamples are found and eliminated in each iteration, thus indicating a higher search efficiency. Fine-tuning time is an overall evaluation of the number of fine-tuning iterations and time consumption of each iteration, the latter of which largely depends on the time consumption of counterexample search. Fine-tuning time also counts the time of all failed verifications, i.e., all verifications except the last one. When the number of verifications exceeds one, all verification fails except the last one. A failed verification means that counterexamples still exist, but adversarial training can no longer find them. Therefore, more verifications also mean that counterexample search is less inefficient. Note that calling verification does not increase fine-tuning iterations because it is a required step of each iteration to check whether verification should be called and call it when necessary.

For verification efficiency, we count the time of the final verification that proves the feasible region conditions hold. This time plus the fine-tuning time equals the total synthesis time of the value function.

For the size of feasible region, we compute a metric called the true feasible rate (TFR), which is defined as the proportion of states identified as feasible in all feasible states. In practice, TFR is approximately computed on a certain number of states randomly sampled in the state space. To determine whether a state is feasible, we check a finite-length trajectory starting from it. The state is considered feasible if there is no constraint violation in the trajectory. The trajectory length is set to 100 for all tasks, which is enough to give correct feasibility results in most cases. For Robot Dog, He et al. [18] synthesized a neural value function without verification and used it as a safety filter in a real-world robot dog locomotion task. We compare the neural value functions synthesized using their method and our framework to demonstrate the necessity and effectiveness of our framework.

5.3 Synthesis Results

Our proposed framework successfully synthesized neural HJ reachability value functions on all nine tasks in Cersyve-9, and the results are shown in Table 3. TFRs on most tasks are greater than 0.8, and the lowest TFR is above 0.4, indicating that the synthesized value networks represent non-trivial feasible regions. As state dimension increases, it generally requires more fine-tuning iterations and time to synthesize a verified value network. This is because searching for counterexamples becomes more difficult in higher-dimensional spaces. The number of verifications also shows a similar increase with state dimension. Another observation is that systems with nonlinear dynamics generally require more verification time. This is because nonlinear dynamics results in more linear segments of $\mathrm{M}_{\mathrm{inv}}$ defined in (17) for forward invariance verification.



(a) TFR.

(b) Cumulative counterexamples.

Fig. 5. TFR and cumulative number of found counterexamples during fine-tuning in Robot Arm. The dashed gray lines stand for the iterations at which verification is called.

To understand the effect of fine-tuning on the value network, we plot the changing curves of TFR and the cumulative number of found counterexamples during fine-tuning in Robot Arm in Figure 5. At an early stage of fine-tuning, TFR decreases quickly, and counterexamples increase quickly. In each iteration, the value network is updated on a large number of counterexamples, excluding many of them from the zero-sublevel set. Verification cannot be called at this stage because there are many counterexamples found in every iteration. After 10K iterations, counterexamples can hardly be found in each iteration, and therefore, verification starts to be called frequently. At this stage, the value network is updated on only a few counterexamples, mostly found by verification, in each iteration. As a result, the change in TFR is very small. This stage continues until the last verification proves the feasible region conditions hold and returns a valid value network.

To demonstrate how counterexamples are eliminated through fine-tuning, we visualize the boundary of the zero-sublevel set and counterexamples before fine-tuning and the boundary after fine-tuning in Double Integrator in Figure 6. Before fine-tuning, there are many counterexamples near the boundary of the zero-sublevel set. These counterexamples leave the zero-sublevel set in one step, violating the forward invariance condition and making the set not a valid feasible region. This invalidity is also confirmed by the fact that the pre-trained region is larger than the true region, which is impossible for a valid feasible region. After fine-tuning, all counterexamples are eliminated, and the zero-sublevel set shrinks into a valid feasible region slightly smaller than the true region.

To demonstrate the necessity and effectiveness of our method, we visualize the value trajectories and heatmaps of the value networks synthesized by ABS and our method in Robot Dog, as shown in Figure 7. We choose an initial state that leads to a constraint violation and compare the values of the two networks on the trajectory. The value network of ABS is negative in the initial state, and gradually increases to positive values along the trajectory. This means that the state starts from inside the zero-sublevel set but goes out eventually, indicating that the zero-sublevel set of their value network is not a valid feasible region because it violates the forward invariance condition. In contrast, our value network consistently outputs positive values on the whole trajectory, indicating that it correctly excludes the states from its zero-sublevel set. Comparing the heatmaps of the two value networks, we can see that our network moves the infeasible region to the upper left. This excludes infeasible

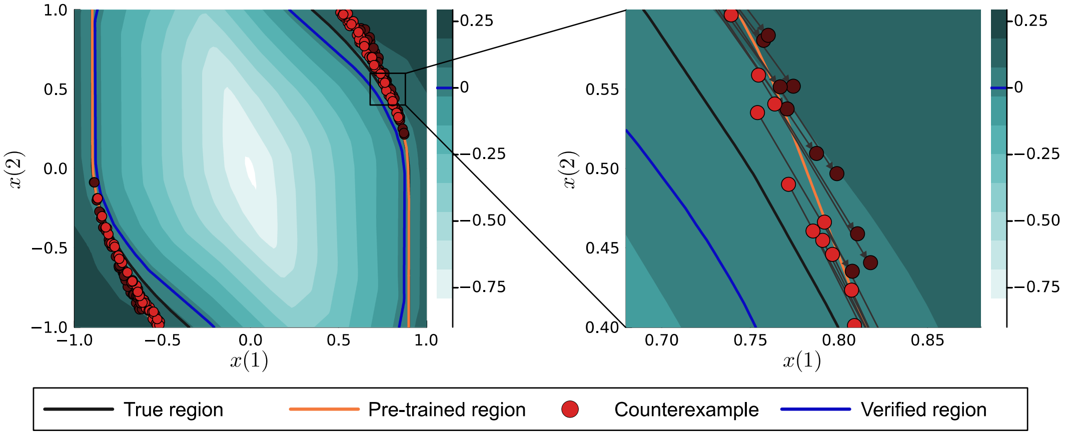


Fig. 6. Regions and counterexamples in Double Integrator. The pre-trained and verified regions are zero-sublevel sets of the value networks before and after fine-tuning, respectively. The heatmap shows the contours of the value network after fine-tuning. The lighter red dots are counterexamples before fine-tuning, and the darker red dots are their next states.

states in the upper left from the zero-sublevel set and includes more feasible states in the lower right into the set. As a result, the zero-sublevel set becomes a valid feasible region without a significant reduction in its size.

5.4 Comparison Studies

In this subsection, we demonstrate the effectiveness of the three proposed techniques, i.e., APA, BGB, and ESR, by comparing them with several existing methods that aim to solve similar problems.

5.4.1 Neural Network Regularization. We compare APA with two existing neural network regularization methods, weight decay (WD) and SNR [38], to study its effectiveness in reducing verification time. We compare these methods from three aspects: number of linear segments, network performance, and verification time. First, we show the relationship between the number of linear segments and verification time. Then, we compare the number of linear segments and network performance of the three regularization methods. Finally, we compare the verification times of the three methods on all tasks in Cersyve-9.

Due to the branch-and-bound solving mechanism of MILP, verification time is closely related to the number of linear segments of neural networks [39]. We use a sampling-based method to estimate the number of linear segments of a neural network. Specifically, we uniformly sample a certain number of states in the state space, compute the neural network's activation pattern on each state, and count the number of unique activation patterns. This gives us an underestimate of the number of linear segments, and this estimate becomes more accurate as the number of samples increases. Figure 8(a) shows the relationship between the estimated number of linear segments and the number of samples. The two have a linear relationship when the number of samples is small. As the number of samples increases, the growth rate of linear segments decreases. Theoretically, an infinite number of samples will give an accurate number of linear segments. We use 10^6 samples in our experiments to balance estimation accuracy and computational complexity. Although this results in an underestimate, it reflects the relative number of linear segments of different methods, which is informative for comparing their verification times. We visualize the relationship between the number of linear segments and verification time in Figure 8(b).

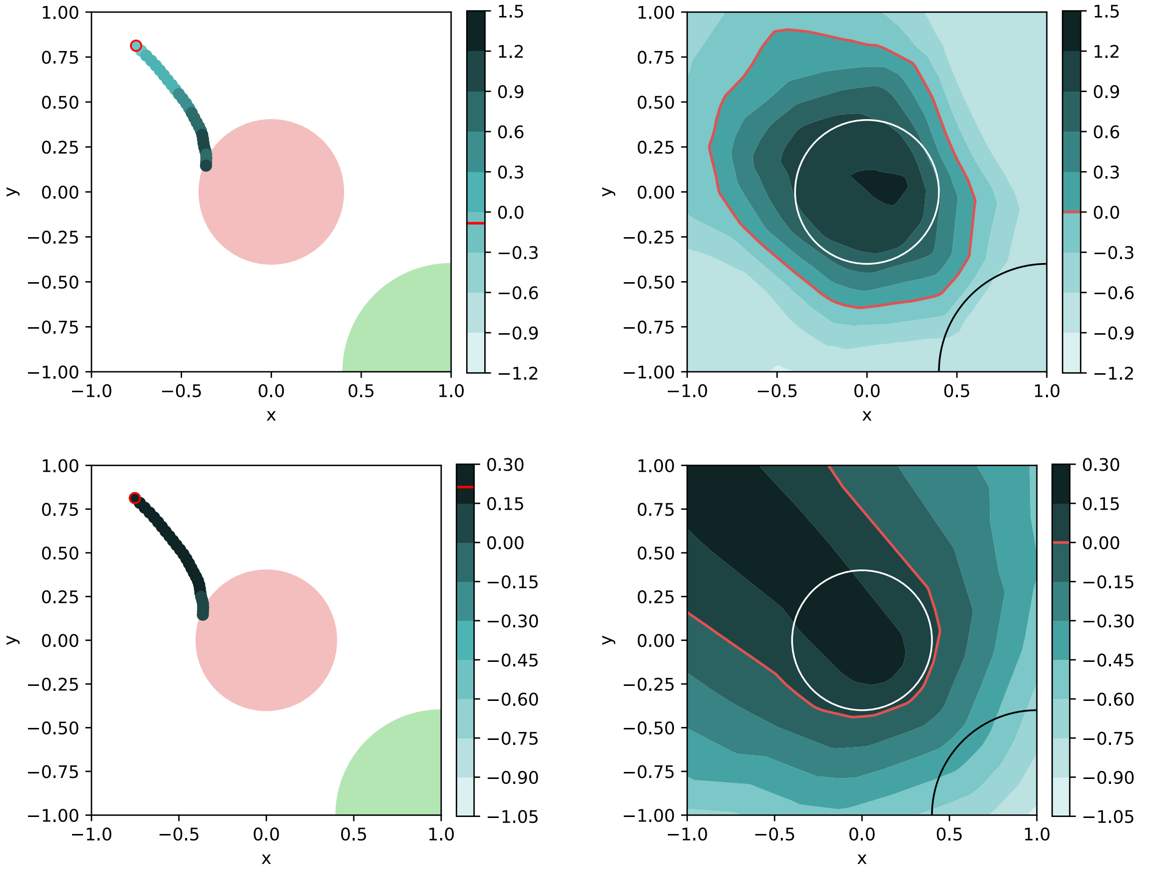
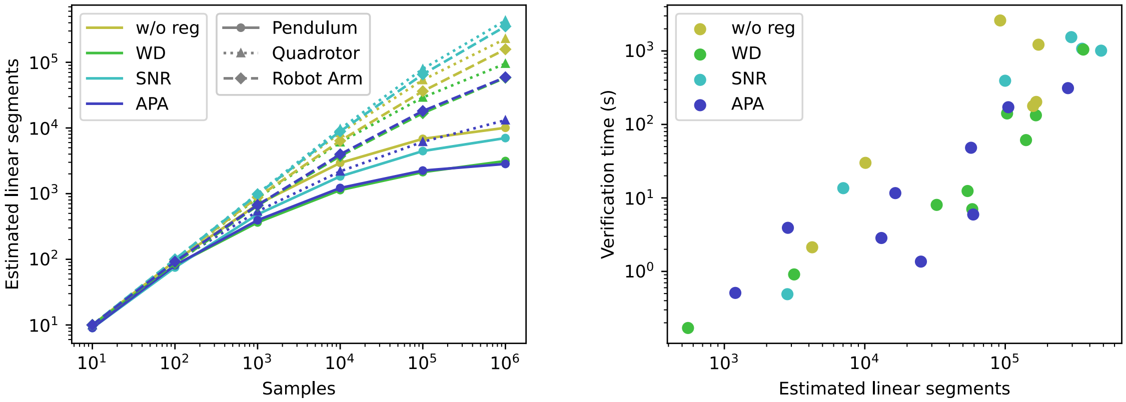


Fig. 7. Value trajectories and heatmaps of value networks synthesized by ABS (upper) and our method (lower) in Robot Dog. In the left two figures, the red circle in the middle is an obstacle, and the green circle in the bottom right corner (a quarter shown) is the goal. The two trajectories start from the same initial state (marked with a red circle) and are sampled by the same policy. They are both truncated at a constraint-violating state.

It shows that the two are approximately linearly related, which is consistent with the branch-and-bound MILP solving mechanism. This allows us to approximately compare the verification times of different networks by comparing their number of linear segments without actually solving verification problems.

We compare the number of linear segments of dynamics networks and value networks trained with different regularization methods, as shown in Figure 9. Since regularization usually sacrifices the performance of neural networks, we also compare the performance of different regularization methods. For dynamics networks, we compute the MSE on a test dataset for performance metrics. For value networks, we compute TFR after fine-tuning (after the network is successfully verified) for performance metrics. For a fair comparison, we use the same dynamics network trained with APA to train value networks in each task. Figure 9(a) shows that APA reduces linear segments of dynamics networks by about five times compared with no regularization, while WD and SNR both increase linear segments instead. Moreover, APA has a much lower MSE compared with WD and SNR. Figure 9(b) shows that both APA and WD significantly reduce linear segments of value networks, and APA



(a) Linear segment estimate.

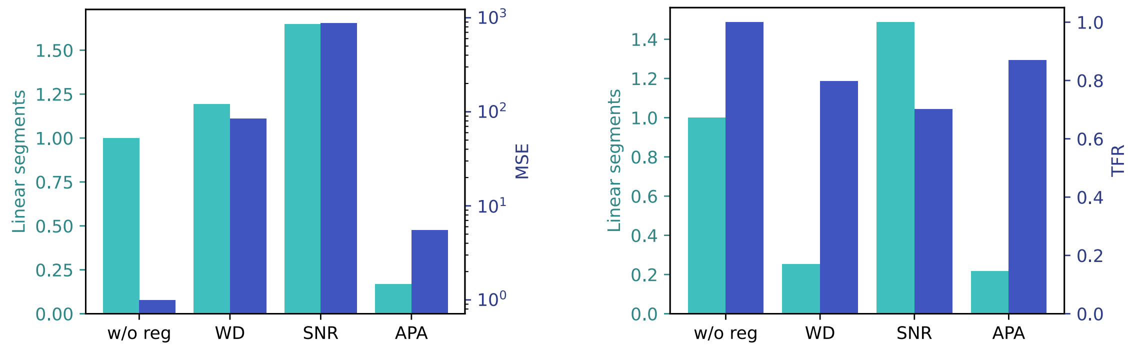
(b) Linear segments vs. verification time.

Fig. 8. Estimated number of linear segments and its relationship with verification time. In Figure (b), dots with the same color represent the same method in different tasks. For a thorough comparison of verification times of different regularization methods on each task, see Table 4.

brings a greater reduction of about four times. SNR still results in increased linear segments of value networks. In addition, APA has a higher TFR compared with WD and SNR, indicating larger feasible regions. These results indicate that APA is the most effective in reducing linear segments with minimum performance sacrifice. This is attributed to the appropriate design of the APA penalty, which only takes effect when the signs of pre-activation values of neighboring states are different. In contrast, WD and SNR are not directly targeted at making activation patterns consistent. They penalize the network parameters at all times, resulting in large performance sacrifices and inefficiency in reducing linear segments.

The regularization strength of APA depends on the coefficient α_{APA}, which trades off between the number of linear segments and neural network performance. We train dynamics and value networks under different values of α_{APA} and visualize the results in Figure 10. Figure 10(a) shows that the number of linear segments of dynamics network quickly decreases as α_{APA} increases from 0 to 10^{-3} and continues to decrease steadily as α_{APA} increases from 10^{-3} to 10^{-1}. On the other hand, MSE also increases as α_{APA} increases, and its increasing rate becomes faster. An appropriate choice of α_{APA} for dynamics network should be around 10^{-3} to 10^{-2}, which balances the number of linear segments and MSE. Figure 10(b) shows that both the number of linear segments of value network and TFR decreases as α_{APA} increases. The decrease rate of linear segments is relatively stable under different values of α_{APA}. The decrease rate of TFR is small at first and gradually increases as α_{APA} increases. An appropriate choice of α_{APA} for value network should be around 10^{-4}.

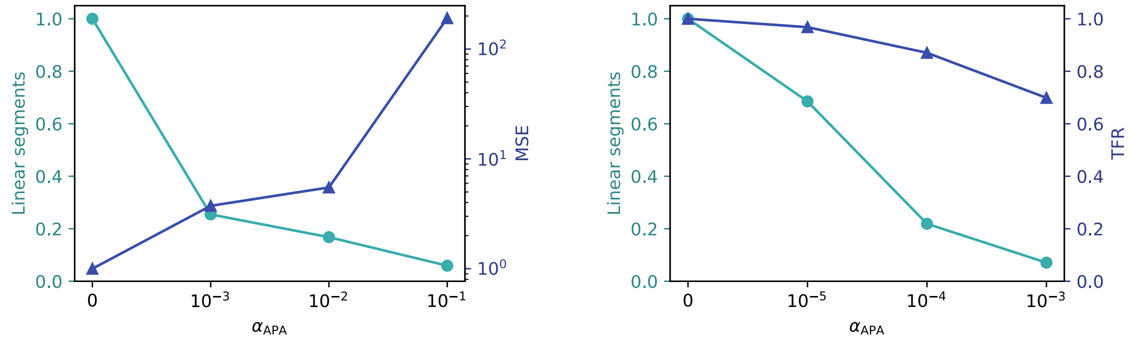
We compare the verification time of different regularization methods in Table 4. It shows that APA has the shortest verification time overall, especially in high-dimensional tasks. In some tasks, such as Unicycle, Lane Keep, and Quadrotor, APA reduces the verification times by more than 100 times compared with no regularization. Note that without regularization, verification may take much longer than the time limit (2 hours) in high-dimensional tasks. The acceleration of verification brought about by APA greatly improves the scalability of our synthesis framework, enabling it to solve higher-dimensional tasks. The superiority of APA is due to its effectiveness in reducing linear segments of both dynamics and value networks. WD also significantly reduces the verification time compared with no regularization because it reduces linear segments of value networks. However, the



(a) Dynamics networks.

(b) Value networks.

Fig. 9. Number of linear segments and performance of dynamics networks and value networks trained with different regularization methods. The results of dynamics networks are averaged over five tasks with nonlinear dynamics. For each task, all scores are normalized by dividing by those without regularization. The results of value networks are averaged on all tasks except Cart Pole and Robot Dog, where synthesis failed without regularization.



(a) Dynamics networks.

(b) Value networks.

Fig. 10. Number of linear segments and performance of dynamics networks and value networks trained under different APA coefficients. The results are averaged on the same tasks as in Figure 9. All scores are normalized by dividing by those of α_{APA}=0.

acceleration of WD is not so significant as that of APA in most nonlinear tasks because it cannot reduce linear segments of dynamics networks. In contrast, SNR has longer verification times than no regularization, which is consistent with the fact that it increases linear segments of both dynamics networks and value networks. SNR performs poorly because it is designed to increase the robustness of a neural network under disturbances, and the results show that this robustness-oriented objective does not always align with the objective of reducing the number of linear segments.

5.4.2 Counterexample Search. We compare our counterexample search method, BGB, with two existing search methods: projected boundary search (PBS) proposed by Liu et al. [26] and PGD with standard backtracking

Table 4. Verification time (in seconds) of different regularization methods. The dynamics networks (for nonlinear tasks) and value networks in each column are trained with the same regularization method. “Timeout" means that fine-tuning exceeds the time limit (2 hours). “MaxIter" means that fine-tuning exceeds the iteration limit (100k).

Task	w/o reg	WD	SNR	APA
Double Integrator	2.1	**0.2**	0.5	0.5
Pendulum	30.0	**0.91**	13.58	3.9
Unicycle	2597.5	12.42	393.3	**11.6**
Lane Keep	202.4	8.0	1540.9	**1.4**
Quadrotor	Timeout	141.01	Timeout	**2.9**
Cart Pole	Timeout	**133.18**	1009.1	171.5
Point Mass	1216.6	61.2	Timeout	**48.2**
Robot Arm	178.3	7.0	1071.8	**5.5**
Robot Dog	Timeout	1043.9	MaxIter	**311.9**

Table 5. Number of fine-tuning iterations and fine-tuning time of different counterexample search methods.

Task	FT iter (k)			FT time (s)		
	PBS	PGD-B	BGB	PBS	PGD-B	BGB
Double Integrator	1.6	**1.2**	**1.2**	**25.3**	50.1	28.5
Pendulum	1.5	**1.1**	**1.1**	**31.3**	55.4	31.4
Unicycle	MaxIter	54.1	**6.9**	MaxIter	3963.6	**175.0**
Lane Keep	41.0	33.1	**5.9**	916.2	2370.0	**184.7**
Quadrotor	21.8	3.1	**1.6**	578.3	223.7	**55.3**
Cart Pole	43.8	62.6	**2.4**	1276.8	4590.0	**83.3**
Point Mass	MaxIter	35.1	**8.6**	MaxIter	3199.2	**321.7**
Robot Arm	MaxIter	MaxIter	**25.2**	MaxIter	MaxIter	**853.8**
Robot Dog	MaxIter	51.5	**6.3**	MaxIter	3436.2	**333.7**

(PGD-B), to study its search efficiency. We count the number of fine-tuning iterations and fine-tuning times of the three search methods, as shown in Table 5. Results show that BGB has the smallest number of fine-tuning iterations and fine-tuning time among the three methods, indicating that it has the highest counterexample search efficiency. PBS has the lowest search efficiency, exceeding the iteration limit on most high-dimensional tasks. This is because projecting the state to the boundary of feasible region in every step is unnecessary and significantly harms search efficiency[3]. We need the state to be close to the boundary only at the final step, not at all intermediate steps. PGD-B also has lower search efficiency than BGB because standard backtracking can only search toward but not along the boundary, making it easy to get stuck near the boundary.

5.4.3 Feasible Region Regularization. We compare our feasible region regularization method, ESR, with RSR [10, 26] and no regularization to study its effectiveness in enlarging feasible regions. Any feasible region regularization method will make fine-tuning harder because it inevitably includes some infeasible states into the zero-sublevel

[3] Projecting the state to the boundary of feasible region in every step is unnecessary not only for HJ reachability but also for other safety certificates such as CBF and CLF, at least in discrete-time systems, because the feasible region conditions are the same for all safety certificates. The projection may become necessary in continuous-time systems where counterexamples must be exactly on the boundary.

Table 6. TFR and number of fine-tuning iterations of different feasible region regularization methods.

Task	TFR			FT iter (k)		
	w/o reg	RSR	ESR	w/o reg	RSR	ESR
Double Integrator	0.897	**0.960**	0.940	**1.1**	1.2	1.2
Pendulum	0.856	0.952	**0.962**	**1.0**	**1.0**	**1.0**
Unicycle	0.671	0.907	**0.911**	**1.2**	3.9	6.9
Lane Keep	0.651	0.693	**0.750**	3.5	**1.6**	5.9
Quadrotor	0.872	0.901	**0.906**	2.2	2.3	**1.6**
Cart Pole	0.014	0.207	**0.404**	4.1	14.5	**2.4**
Point Mass	0.180	0.549	**0.594**	**1.6**	7.5	8.6
Robot Arm	0.000	**0.405**	0.403	**3.5**	79.3	25.2
Robot Dog	0.659	0.865	**0.872**	**4.0**	12.2	6.3

set. To evaluate the negative impact on fine-tuning, we not only compute the TFR of the value networks but also count the number of fine-tuning iterations of each regularization method, as shown in Table 6. It shows that ESR has the highest TFR on almost all tasks, significantly increasing TFR compared with no regularization, especially on high-dimensional tasks. TFR of no regularization becomes smaller as the state dimension increases, indicating that fine-tuning tends to mistakenly exclude feasible states from the zero-sublevel set, resulting in feasible region shrinkage. RSR also increases TFR compared with no regularization, but it is not so effective as ESR, and its number of fine-tuning iterations is not less than ESR. This is because RSR randomly pushes all states into the zero-sublevel set, which will mistakenly include more infeasible states than ESR, resulting in lower regularization efficiency and a greater negative impact on fine-tuning.

5.4.4 Ablation Study. We perform ablation studies to show how the proposed three techniques contribute to the reduction of overall synthesis time and the increase of TFR, and the results are shown in Figure 11.

First, we test a baseline algorithm called Vanilla that directly minimizes MSE without neural network regularization in pre-training, uses PGD-B to search counterexamples, and performs fine-tuning without feasible region regularization. Results show that Vanilla fails to synthesize value functions on three higher-dimensional nonlinear tasks, i.e., Cart Pole, Point Mass, and Robot Dog. Moreover, it also fails on Robot Arm because the TFR is zero, i.e., the zero-sublevel set of the value function shrinks to an empty set.

Next, we add APA in pre-training and keep the adversarial training part unchanged. Results show that APA significantly reduces synthesis time on almost all tasks, especially higher-dimensional ones. The comparison of synthesis time on Robot Arm is meaningless because all algorithms fail to synthesize a non-trivial value function except the last one that uses all three techniques. APA's reduction of synthesis time is mainly attributed to its acceleration of verification, not only the last verification that proves hold but also intermediate failed verifications.

Then, we add BGB for counterexample search and keep the fine-tuning loss unchanged. Results show that BGB further reduces synthesis time on all tasks and does not cause significant changes in TFR. BGB's reduction of synthesis time is mainly attributed to its acceleration of counterexample search, which results in fewer fine-tuning iterations.

Finally, we add ESR to fine-tuning loss, obtaining the complete version of our algorithm. Results show that ESR substantially increases TFR on almost all tasks, especially Cart Pole and Robot Arm, where other algorithms fail or almost fail to synthesize non-trivial value functions. Although the synthesis times of the complete algorithm increase compared with APA+BGB on some tasks, it still achieves a large acceleration compared with Vanilla. Except for the first two lower-dimensional tasks, the acceleration compared with Vanilla is close to or more than

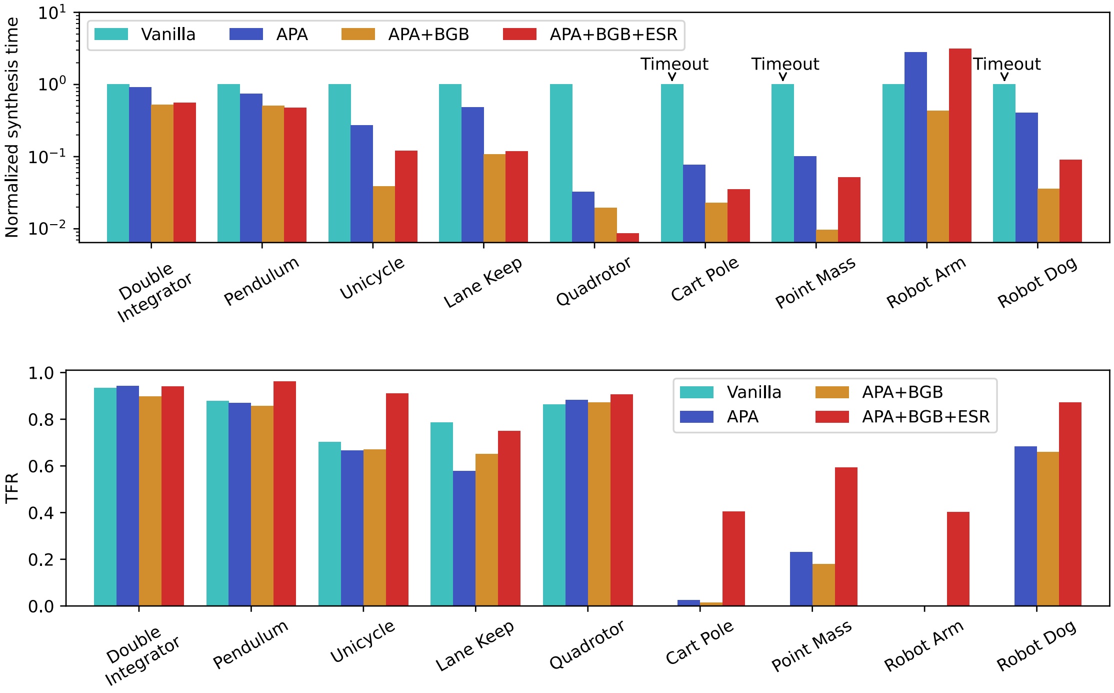


Fig. 11. Ablation study of three techniques with respect to synthesis time and TFR. The normalized synthesis time is the synthesis time, i.e., fine-tuning time plus verification time, of each algorithm divided by that of Vanilla. The "Timeout" annotation on top of the bars means the corresponding experiments exceed the time limit (2 hours), and we use the time limit for normalization in these tasks.

10 times on all tasks, and the acceleration on Quadrotor reaches about 100 times. Note that Vanilla exceeds the time limit on three tasks, where the acceleration of our techniques could be much greater than that shown in the figure. These results indicate that APA and BGB significantly reduce synthesis time, ESR substantially increases TFR, and these three techniques together significantly improve the scalability of our framework.

6 Conclusion

This paper proposes a scalable framework for formally synthesizing verified neural HJ reachability value functions. The framework consists of three stages: pre-training, adversarial training, and verification-guided training. We propose three techniques that significantly improve the scalability of our framework: boundary-guided backtracking (BGB) to accelerate counterexample search, entering state regularization (ESR) to enlarge feasible regions, and activation pattern alignment (APA) to accelerate MILP-based verification. We also provide a neural safety certificate synthesis and verification benchmark called Cersyve-9, including nine commonly used safe control tasks. Our framework successfully synthesizes verified neural value functions on all tasks in our benchmark. Extensive experiments show that the three proposed techniques exhibit superior scalability and efficiency compared with existing methods. While our experiments mainly focus on the synthesis side, the proposed benchmark could also foster additional study in the verification community on scaling up verification algorithms with respect to these unique types of problems.

While our proposed framework improves scalability for reachability analysis, it is still limited to state dimensions up to 6. Future directions for extending to higher-dimensional systems include using more advanced verification algorithms with parallel computation and exploring more efficient adversarial training or certified training methods to eliminate counterexamples. In addition, our framework is only evaluated in simulation in this work. To bring this framework to the real world, we will need to first solve the robust verification problem that accounts for model uncertainty as discussed in Section 4.5, which will be left for future work.

Acknowledgments

This work is in part supported by the National Science Foundation under Grant No. 2144489. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Alessandro Abate, Daniele Ahmed, Alec Edwards, Mirco Giacobbe, and Andrea Peruffo. 2021. FOSSIL: a software tool for the formal synthesis of lyapunov functions and barrier certificates using neural networks. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*. 1–11.

[2] Alessandro Abate, Daniele Ahmed, Mirco Giacobbe, and Andrea Peruffo. 2020. Formal synthesis of Lyapunov neural networks. *IEEE Control Systems Letters* 5, 3 (2020), 773–778.

[3] Anayo K. Akametalu, Shromona Ghosh, Jaime F. Fisac, Vicenc Rubies-Royo, and Claire J. Tomlin. 2024. A Minimum Discounted Reward Hamilton–Jacobi Formulation for Computing Reachable Sets. *IEEE Trans. Automat. Control* 69, 2 (2024), 1097–1103. https://doi.org/10.1109/TAC.2023.3327159

[4] Mahathi Anand and Majid Zamani. 2023. Formally verified neural network control barrier certificates for unknown systems. *IFAC-PapersOnLine* 56, 2 (2023), 2431–2436.

[5] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. 2017. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2242–2253.

[6] Somil Bansal and Claire J Tomlin. 2021. Deepreach: A deep learning approach to high-dimensional reachability. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1817–1824.

[7] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. 2017. *Predictive control for linear and hybrid systems*. Cambridge University Press.

[8] Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T Johnson, and Changliu Liu. 2023. First three years of the international verification of neural networks competition (VNN-COMP). *International Journal on Software Tools for Technology Transfer* 25, 3 (2023), 329–339.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym.

[10] Ya-Chien Chang, Nima Roohi, and Sicun Gao. 2019. Neural lyapunov control. *Advances in neural information processing systems* 32 (2019).

[11] Mo Chen, Sylvia Herbert, and Claire J Tomlin. 2016. Fast reachable set approximations via state decoupling disturbances. In *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 191–196.

[12] Shaoru Chen, Lekan Molu, and Mahyar Fazlyab. 2024. Verification-Aided Learning of Neural Network Barrier Functions with Termination Guarantees.

[13] Hongkai Dai, Benoit Landry, Lujie Yang, Marco Pavone, and Russ Tedrake. 2021. Lyapunov-stable neural-network control.

[14] Jérôme Darbon, Gabriel P Langlois, and Tingwei Meng. 2020. Overcoming the curse of dimensionality for some Hamilton–Jacobi partial differential equations via neural network architectures. *Research in the*

Mathematical Sciences 7, 3 (2020), 20.

[15] Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*. Springer, 269–286.

[16] Jaime F Fisac, Neil F Lugovoy, Vicenç Rubies-Royo, Shromona Ghosh, and Claire J Tomlin. 2019. Bridging hamilton-jacobi safety analysis and reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 8550–8556.

[17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples.

[18] Tairan He, Chong Zhang, Wenli Xiao, Guanqi He, Changliu Liu, and Guanya Shi. 2024. Agile But Safe: Learning Collision-Free High-Speed Legged Locomotion.

[19] Kai-Chieh Hsu, Haimin Hu, and Jaime F Fisac. 2023. The safety filter: A unified view of safety-critical control in autonomous systems. *Annual Review of Control, Robotics, and Autonomous Systems* 7 (2023).

[20] Kai Chieh Hsu, Vicenç Rubies-Royo, Claire J Tomlin, and Jaime F Fisac. 2021. Safety and Liveness Guarantees through Reach-Avoid Reinforcement Learning. In *17th Robotics: Science and Systems, RSS 2021*. MIT Press Journals.

[21] Hanjiang Hu, Yujie Yang, Tianhao Wei, and Changliu Liu. 2025. Verification of Neural Control Barrier Functions with Symbolic Derivative Bounds Propagation. In *Proceedings of The 8th Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 270)*. PMLR, 1797–1814.

[22] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. 2019. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 169–178.

[23] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images.

[24] Yann LeCun, Corinna Cortes, Chris Burges, et al. 2010. MNIST handwritten digit database.

[25] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. 2021. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization* 4, 3-4 (2021), 244–404.

[26] Simin Liu, Changliu Liu, and John Dolan. 2023. Safe control under input limits with neural control barrier functions. In *Conference on Robot Learning*. PMLR, 1970–1980.

[27] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks.

[28] Ian M Mitchell. 2007. Comparing forward and backward reachability as tools for safety analysis. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 428–443.

[29] Ian M Mitchell. 2008. The flexible, extensible and efficient toolbox of level set methods. *Journal of Scientific Computing* 35 (2008), 300–329.

[30] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. 2018. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 7559–7566.

[31] Michael P Owen, Adam Panken, Robert Moss, Luis Alvarez, and Charles Leeper. 2019. ACAS Xu: Integrated collision avoidance and detect and avoid capability for UAS. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE, 1–10.

[32] Andrea Peruffo, Daniele Ahmed, and Alessandro Abate. 2021. Automated and formal synthesis of neural barrier certificates for dynamical models. In *International conference on tools and algorithms for the construction and analysis of systems*. Springer, 370–388.

[33] Manan Tayal, Hongchao Zhang, Pushpak Jagtap, Andrew Clark, and Shishir Kolathaya. 2024. Learning a Formally Verified Control Barrier Function in Stochastic Environment.

[34] Vincent Tjeng, Kai Xiao, and Russ Tedrake. 2017. Evaluating robustness of neural networks with mixed integer programming.

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

[36] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems* 34 (2021), 29909–29921.

[37] Xinyu Wang, Luzia Knoedler, Frederik Baymler Mathiesen, and Javier Alonso-Mora. 2023. Simultaneous synthesis and verification of neural control barrier functions through branch-and-bound verification-in-the-loop training.

[38] Tianhao Wei, Ziwei Wang, Peizhi Niu, Abulikemu Abuduweili, Weiye Zhao, Casidhe Hutchison, Eric Sample, and Changliu Liu. 2024. Improve Certified Training with Signal-to-Noise Ratio Loss to Decrease Neuron Variance and Increase Neuron Stability.

[39] Laurence A Wolsey. 2020. *Integer programming*. John Wiley & Sons.

[40] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. 2020. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers.

[41] Dongjie Yu, Wenjun Zou, Yujie Yang, Haitong Ma, Shengbo Eben Li, Yuming Yin, Jianyu Chen, and Jingliang Duan. 2023. Safe model-based reinforcement learning with an uncertainty-aware reachability certificate.

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

[43] Hongchao Zhang, Junlin Wu, Yevgeniy Vorobeychik, and Andrew Clark. 2024. Exact verification of relu neural control barrier functions. *Advances in Neural Information Processing Systems* 36 (2024).

Received 31 July 2024; revised 27 May 2025; accepted 18 June 2025