Learning Verifiable Control Policies Using Relaxed Verification

Puja Chaudhury, Alexander Estornell, Michael Everett

***Abstract*— To provide safety guarantees for learning-based control systems, recent work has developed formal verification methods to apply after training ends. However, if the trained policy does not meet the specifications, or there is conservatism in the verification algorithm, establishing these guarantees may not be possible. Instead, this work proposes to perform verification throughout training to ultimately aim for policies whose properties can be evaluated throughout runtime with lightweight, relaxed verification algorithms. The approach is to use differentiable reachability analysis and incorporate new components into the loss function. Numerical experiments on a quadrotor model and unicycle model highlight the ability of this approach to lead to learned control policies that satisfy desired reach-avoid and invariance specifications.**

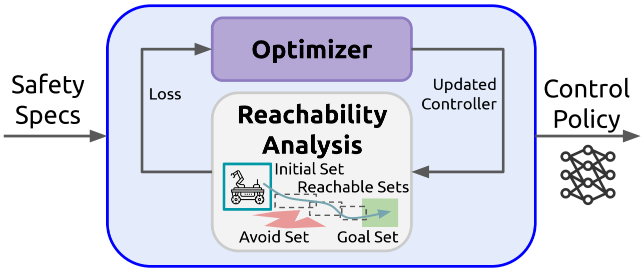


Fig. 1: Learning verifiable control policies. The objective is to synthesize a NN control policy guided by a safety specifications. At each training iteration, loss terms based on reachability analysis are used to update the NN parameters.

I. Introduction

Accounting for safety requirements remains a challenge in learning-based control, which is essential in applications such as aerospace and autonomous driving. Motivated by this issue, recent work has developed formal verification techniques for neural networks (NNs) [1]–[9] and NN-controlled dynamical systems [9]–[21]. For example, these techniques can be used to prove stability, calculate regions of attraction, or estimate forward and backward reachable sets, as part of a comprehensive safety analysis.

However, due to the computational cost of formal verification (e.g., exact verification of input-output properties of ReLU NNs is NP-Complete [7]), these techniques are usually applied *after* learning is complete. If the resulting system does not satisfy the safety specifications, naturally, no sound verification algorithm would be able to prove that it does. And even if the resulting system does satisfy safety specs, it may not be tractable to perform complete verification [9], [22], [23], especially for run-time monitoring throughout operation. Meanwhile, the looseness introduced by sound but incomplete (fast) verification algorithms, such as [1], [12], [14], [23], may still prevent obtaining a proof.

Instead, this work focuses on guiding the learning process to meet performance objectives and satisfy safety specifications by performing verification *during* each training iteration. While this is common in training NNs in isolation (e.g., for image classification [24]), relatively fewer have considered synthesizing NN controllers with safety and performance guarantees (e.g., [25]–[35]), or more specifically, considered verification while training NN control policies. For example, several existing methods propose to simultaneously learn a policy and certificate (e.g., [29]–[32]) with counterexample-guided training. A limitation of counterexample-guided training is that it is not obvious which and how many points to sample at each iteration. Alternatively, [33] proposed to add loss terms based on forward reachable sets, with numerical estimates of gradients used to perform policy updates. [34], [35] further extended this idea using differentiable reachable set estimates for reinforcement learning. Building on these advances, this work investigates ways to synthesize useful control policies strictly using the differentiable reachable set bounds, without relying on an external reward signal from the environment.

The primary contribution of this work is an approach for guiding NN control policy training to encourage safety specification satisfaction, illustrated in Fig. 1. The approach uses CROWN [1] to compute reachable set estimates that are differentiable with respect to the controller parameters. This enables defining reachability-related terms in the loss function, such as robustly reaching a goal region, avoiding obstacles, or forming other invariant sets. Numerical experiments on unicycle and 6D quadrotor models demonstrate that the policies can be trained quickly and the resulting systems not only satisfy the specifications, but they also can be verified quickly, which is important for run-time monitoring, where the specifications may change throughout operation.

II. Preliminaries

A. Problem Statement

For a state space $\mathcal{X} \subseteq \mathbb{R}^{n_x}$ and control space $\mathcal{U} \subseteq \mathbb{R}^{n_u}$, this paper aims to find a feedback control policy $\pi_\theta : \mathcal{X} \rightarrow \mathcal{U}$, parameterized by θ, such that a system's closed-loop dynamics satisfy reach-avoid properties. In particular, denote the discrete-time closed-loop dynamics as:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \pi_\theta(\mathbf{x}_t)), \qquad (1)$$

Authors are with Northeastern University, Boston, MA, USA. e-mail: {chaudhury.p,estornell.a,m.everett}@northeastern.edu

where $\mathbf{x}_t \in \mathcal{X}$ is the system state at time t. When the closed-loop dynamics comprise of NN components (e.g., NN controller, NN dynamics), we will call the system in (1) a *neural feedback loop*, which suggests the controller and/or dynamics may be high-dimensional and nonlinear. It remains challenging to find θ such that a neural feedback loop satisfies complicated reach-avoid properties (e.g., navigating through an environment with many obstacles).

B. Verification of Neural Networks

Much of the recent work on neural feedback loops has focused on analysis. That is, given a *trained* control policy with fixed parameters θ, does the policy meet the reach-avoid specifications? While the *exact* neural feedback loop verification problem can be encoded as a nonlinear program (e.g., mixed-integer program [10] when π uses ReLU activations), computational limits often motivate sound (but incomplete) verifiers, which leverage convex relaxations to obtain linear or semidefinite programs.

Theorem II.1 (Linear Relaxation-based Perturbation Analysis (LiRPA))**.** *Given a function $g : \mathcal{X} \subseteq \mathbb{R}^n \to \mathcal{Y} \subseteq \mathbb{R}^m$, input set $\mathcal{X}' \subseteq \mathcal{X}$, and polytope facets $\mathbf{C} \in \mathcal{R}^{c\times m}$, LiRPA calculates $\mathbf{d} \in \mathbb{R}^c$, which defines polytope outer bounds on the image $g(\mathcal{X}') \subseteq \{\mathbf{y} \in \mathcal{Y} \,|\, \mathbf{C}\mathbf{y} \leq \mathbf{d}\}$.*

The details of how LiRPA calculates $\mathbf{d}$ are provided in [1]. For a one sentence summary, LiRPA computes affine bounds on each primitive in the computation graph, g, that are guaranteed to hold over the input domain to that primitive, then aggregates all of these affine bounds, then concretizes the bounds from function input to output (in closed-form for l_p-ball input domains). The codebases [36], [37] support performing LiRPA on computation graphs with a wide range of primitives that appear in neural networks and dynamical systems (e.g., trigonometric functions, affine transformations, ReLU, sigmoid). Backward CROWN – which we will refer to simply as CROWN – is a type of LiRPA.

C. Verification of Neural Feedback Loops

Moreover, calculating over-approximations of forward reachable sets of a neural feedback loop is a straightforward application of Theorem II.1.

Corollary II.2 ([1], Closed-Loop Reachability Analysis with CROWN)**.** *Given a neural feedback loop f, initial state set $\mathcal{X}_t \subseteq \mathcal{X}$, and polytope facets $\mathbf{C} \in \mathbb{R}^{c\times n}$, the system's next state $\mathbf{x}_{t+1}$ must be in the 1-step reachable set, $R_1(\mathcal{X}_t)$, and its outer bound, $\bar{R}_1(\mathcal{X}_t)$:*

$$R_1(\mathcal{X}_t) \triangleq \{\mathbf{x}_{t+1} \mid \mathbf{x}_{t+1} = f(\mathbf{x}_t, \pi_\theta(\mathbf{x}_t))\} \tag{2}$$

$$\subseteq \{\mathbf{x}_{t+1} \mid \mathbf{C}\mathbf{x}_{t+1} \leq \mathbf{d}\} \triangleq \bar{R}_1(\mathcal{X}_t). \tag{3}$$

Several recent papers explore the trade-offs between computational cost and bound tightness for estimating reachable sets over T timesteps: either by running Corollary II.2 T times iteratively or running Corollary II.2 once over a computation graph that contains T copies of f (e.g., [10], [11], [38], [39]). This paper will use the former approach.

To check whether the system meets the reach-avoid specifications, common approaches are to (a) calculate the reachable sets explicitly and check for intersection/containment with the avoid/reach sets, respectively, or (b) encode the reach-avoid properties as additional layers at the end of the dynamics and check for feasibility.

In this work, we will let $\mathbf{C} = [\mathbf{I}_{n_x \times n_x}, -\mathbf{I}_{n_x \times n_x}]^\top$ to obtain hyperrectangle bounds on the state vector. Therefore, for $\mathbf{C}\mathbf{x} \leq \mathbf{d}$, let $\mathbf{d} = [\bar{\mathbf{x}}, -\underline{\mathbf{x}}]^\top$. We will sometimes refer to reachable set bounds $\bar{\mathcal{R}}$ either as a set, as in (3), or by $\bar{\mathbf{x}}$ and $\underline{\mathbf{x}}$ as the upper and lower (elementwise) bounds on $\mathbf{x}$, respectively.

III. Approach

The controller training process incorporates verification results through a loss function with multiple objectives,

$$\mathcal{L}(\theta) = \sum w_{...}\ \mathcal{L}_{...} \tag{4}$$

where each term encodes an aspect of system performance:

1) *Goal/Obstacle Overlap*: penalizes the volume of (positional) reachable sets that are not within a region, e.g., around the goal position $\mathbf{x}_g$,

$$\mathcal{L}_{\text{overlap}} = \sum_{t=0}^{T} \prod_{i=0}^{n_x-1} \max(\min(\bar{\mathbf{x}}_{t,i}, \mathbf{x}_{g,i} + 0.5) - \max(\underline{\mathbf{x}}_{t,i}, \mathbf{x}_{g,i} - 0.5), 0). \tag{5}$$

2) *Goal-Reaching*: drives the system toward the goal region, by penalizing the distance between the center of the goal region, $\mathbf{x}_g$, and the center of each reachable set,

$$\mathcal{L}_{\text{goal}} = \sum_{t=0}^{T} \| \frac{\bar{\mathbf{x}}_t + \underline{\mathbf{x}}_t}{2} - \mathbf{x}_g \|_2. \tag{6}$$

3) *Bound Volume*: encourages controller parameters such that the system's reachable sets can be bounded tightly (i.e., small volume) by a *relaxed* verification algorithm,

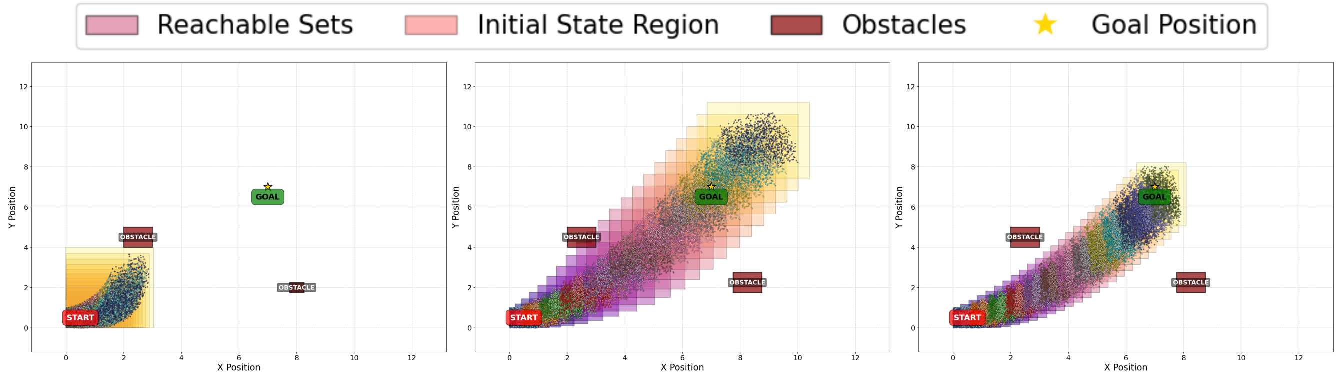
$$\mathcal{L}_{\text{vol}} = \sum_{t=0}^{T} \prod_{i=0}^{n_x-1} \left(\bar{\mathbf{x}}_{t,i} - \underline{\mathbf{x}}_{t,i}\right). \tag{7}$$

4) *Invariance*: encourages establishing an invariant set later in the trajectory, to ensure the system remains sufficiently close to the goal region indefinitely,

$$\mathcal{L}_{\text{inv}} = \sum_{t=t_{\text{inv}}}^{T} \|\bar{\mathbf{x}}_{t+1} - \bar{\mathbf{x}}_t\|_2 + \|\underline{\mathbf{x}}_{t+1} - \underline{\mathbf{x}}_t\|_2, \tag{8}$$

where t_{inv} is the timestep to start encouraging invariance. Without this term, a common issue in using relaxed verification algorithms is that samples from the reachable sets suggest there is likely an invariant set around the goal, but the looseness in the bounds presents difficulty in proving this.

The verification-guided training procedure is summarized in Algorithm 1. After randomly initializing the policy, each training iteration involves computing the reachable set bounds $\bar{\mathcal{R}}_1, \dots, \bar{\mathcal{R}}_T$ for some time horizon using



(a) NN Policy, Loss with Numerical Gradients (b) Affine Policy, Loss with CROWN (c) NN Policy, Loss with CROWN

Fig. 2: Comparison of reachable sets computed by each method for a unicycle model and reach-avoid specification. The policy trained with numerical gradients (left) avoids obstacles but does not reach the goal. The affine policy trained with CROWN loss (middle) reaches the goal and its sampled trajectories avoid the obstacles, but the reachable set bounds collide with an obstacle. The NN policy (right) trained with CROWN loss reaches the goal, and the reachable set bounds are sufficiently tight that they also avoid the obstacles.

Algorithm 1 Verification-In-The-Loop Training

```
Input: Initial states $\mathcal{X}_0$, dynamics $f$, goal $\mathcal{G}$, obstacles $\mathcal{A}_{0:A}$
Output: Optimized parameters, $\theta^*$
 1: $\theta_0 \leftarrow$ randomly initialized parameters
 2: optimizer $\leftarrow$ Adam(lr = 0.0001) [40]
 3: $\bar{\mathcal{R}}_0 \leftarrow \mathcal{X}_0$
 4: while not converged do
 5:     for $t \in \{0, \ldots, T\}$ do
 6:         $\bar{\mathcal{R}}_{t+1} \leftarrow$ CROWN$(f, \bar{\mathcal{R}}_t)$, where $f$ uses $\pi_{\theta_i}$ [1]
 7:     end for
 8:     Evaluate $\mathcal{L}$ via Eqs. (5) to (8)
 9:     $\theta_{i+1} \leftarrow \theta_i -$ optimizer.step$(\nabla_\theta \mathcal{L}(\theta_i))$
10:     Check if verification criteria are satisfied
11: end while
12: return $\theta$
```

CROWN [1] (Line 6). Then, after calculating the terms of the loss function (Line 8), the optimizer takes a step in the controller parameter space (Line 9). This loop continues until a termination condition is reached (e.g., max number of iterations, convergence).

We note that the use of soft penalties as opposed to hard constraints during training could lead to systems that do not necessarily meet the safety specs. In those cases, ideas proposed in [35], such as using n-step recursive calculations of the forward reachable sets, or training different policies for different subsets of $\mathcal{X}_0$, could be beneficial. Nonetheless, the experiments in the next section demonstrate various cases where the modified loss does indeed lead to specification satisfaction.

IV. Results

This section demonstrates the proposed method on several control synthesis tasks with different specifications. First, we show that the differentiable reachability-based approach enables learning a policy for a unicycle to avoid obstacles and reach a goal region, with faster learning and better performance than a prior approach based on numerical gradient estimates [33]. Next, we show that the bound tightness term in the loss function leads to a system whose reachable sets are much tighter, with similar performance on the task, which would enable the use of fast verification methods during runtime. Then, we demonstrate that the invariance term in the loss can lead to an invariant set around the goal region. Finally, we highlight the scalability of the method on a 6D quadrotor model for obstacle avoidance while flying toward a goal region.

A. Obstacle Avoidance: Unicycle

The discrete-time unicycle system evolves according to:

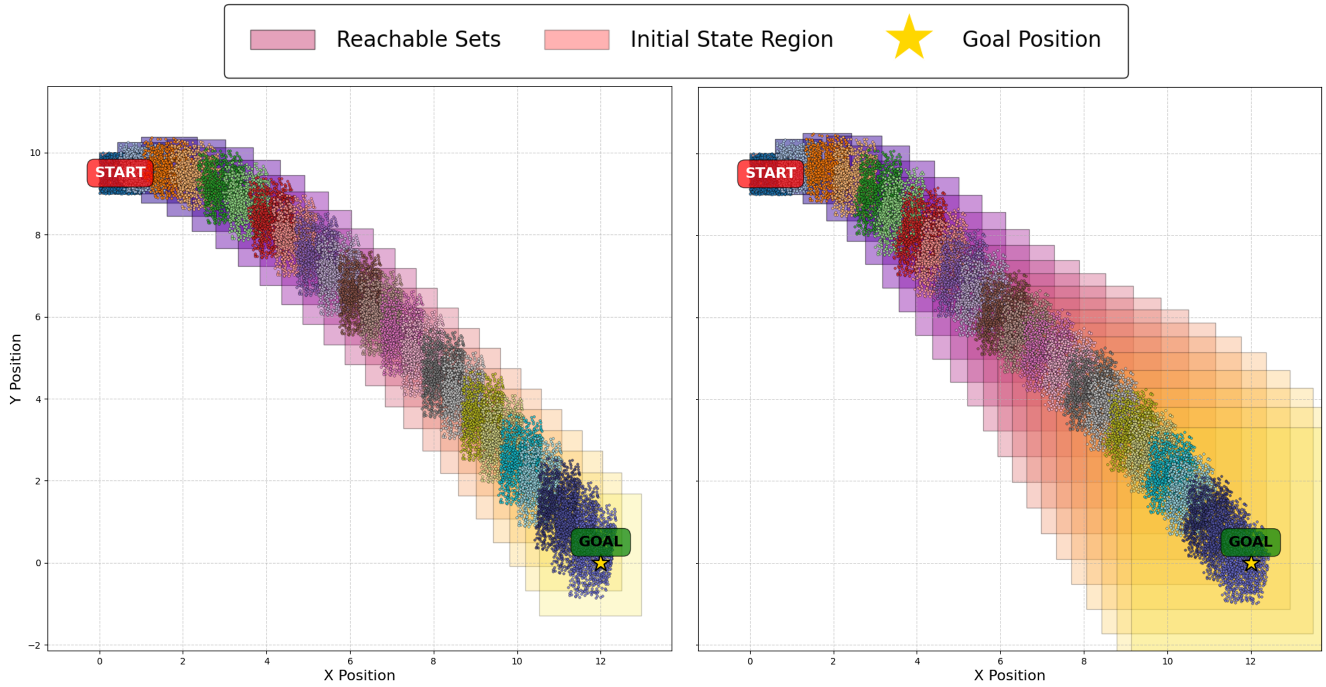
$$\begin{aligned} x_{t+1} &= x_t + v_t \cos(\theta_t) \\ y_{t+1} &= y_t + v_t \sin(\theta_t) \\ \theta_{t+1} &= \theta_t + \omega_t, \end{aligned} \tag{9}$$

where state $\mathbf{x} = [x, y, \theta]$ is composed of position and heading angle, and control inputs $\mathbf{u} = [v, \omega]$ are linear and angular velocity, respectively.

The system must navigate from initial state $\mathbf{x}_0 = (0, 1)$ to a goal region $\mathcal{G}$ centered at $(7, 7)$, while avoiding two obstacles, $\mathcal{A}_0$ centered at $(2, 4)$ and $\mathcal{A}_1$ at $(8, 2)$. Initial state uncertainty is bounded by ± 0.1 in each dimension.

Fig. 2 shows the reachable sets after training with three different methods. On the left (Fig. 2a), an affine control policy $\mathbf{u}_t = \mathbf{k} \cdot \mathbf{x}_t + \mathbf{b}$, i.e., $\theta = [\mathbf{k}, \mathbf{b}]$, where $\mathbf{k}, \mathbf{b} \in \mathbb{R}^3$ was trained using the numerical gradient approximation technique from [33] to update the control parameters during training. While these numerical gradient approaches worked in our experiments replicating the Van der pol oscillator[1] and cruise

[1] Furthermore, for the Van der pol oscillator system (from [33], not shown here), the training time was 187.60 seconds using the numerical gradient approach [33], compared to 44.52 seconds with our proposed method.



(a) With $w_{\text{vol}} = 4$

(b) With $w_{\text{vol}} = 0.05$

Fig. 3: Effect of Bound Volume Loss, $\mathcal{L}_{\text{vol}}$. For the unicycle system with a different start/goal from before, the reachable sets calculated by CROWN after training are much tighter when $w_{\text{vol}} = 4$ (left) than $w_{\text{vol}} = 0.05$ (right), even though the system's performance and true reachable sets are nearly identical. This highlights the benefit of including $\mathcal{L}_{\text{vol}}$ in the training process: it enables obtaining reasonably tight bounds even with a relaxed verification algorithm.

control examples from [33], that method did not perform well on this unicycle system for either an affine policy or NN policy (not shown). Conversely, the proposed method begins to achieve the specification given only an affine policy and performs well with a NN policy.

In the middle (Fig. 2b) is an implementation of the proposed Algorithm 1 with an affine policy, where $\mathbf{k} \in \mathbb{R}^{3\times 2}$ and $\mathbf{b} \in \mathbb{R}^2$. The vehicle reaches the goal (and goes past it) while avoiding the obstacles, but the reachable set bounds calculated via CROWN loose enough to intersect the obstacles. This would prevent an obstacle avoidance proof without a more expensive verifier.

On the right, (Fig. 2c) uses a NN control policy. With this policy, the system successfully reaches the goal region in 24 steps, and neither the "true" reachable set samples and the over-approximations, $\bar{\mathcal{R}}_1, \ldots, \bar{\mathcal{R}}_{24}$ intersect with the avoid sets. The NN control policy is composed of 3 hidden layers, with [16, 32, 16] neurons, ReLU activations, and parameters initialized using scaled uniform distribution, between $\pm 0.1\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}$. Other hyperparameters include $T = 24$, Adam optimizer with learning rate 1e-4, and 20,000 epochs.

For the CROWN methods, the loss function was

$$\begin{aligned}\mathcal{L}(\theta) = w_{\text{goal}}\mathcal{L}_{\text{goal}} + w_{\text{overlap\_obs}}\mathcal{L}_{\text{overlap\_obs}} + \\ w_{\text{overlap\_goal}}\mathcal{L}_{\text{overlap\_goal}} + w_{\text{vol}}\mathcal{L}_{\text{vol}},\end{aligned} \qquad (10)$$

with $w_{\text{goal}} = 8$, $w_{\text{overlap\_danger}} = -15$, $w_{\text{overlap\_goal}} = 20$, $w_{\text{vol}} = 0.5$. $\mathcal{L}_{\text{overlap\_goal}}$ uses (5) as written, and $\mathcal{L}_{\text{overlap\_obs}}$ replaces the goal center with each obstacle center. We note that the loss function for the numerical gradient approach was slightly different and will refer readers to the code for exact implementation details.

B. Bound Tightness

To illustrate the effect of the bound tightness loss term, $\mathcal{L}_{\text{vol}}$ from (7), Fig. 3 shows a unicycle system controlled by a policy trained with $w_{\text{vol}} = 4$ (left), and $w_{\text{vol}} = 0.05$ (right). Both trained policies lead to almost identical system behavior and *true* reachable sets according to the sampled trajectories. However, the bounds shown using CROWN, a relaxed/fast verifier, are much tighter after considering the relaxations throughout training. In other words, it would take much more computational effort (e.g., using branch-and-bound [12], one-shot analysis [11]) to establish bounds of similar tightness for the verification-unaware system. This result is significant because enabling the use of a lightweight verifier to compute reachable set bounds of reasonable tightness at run-time is important for safety-critical control systems.

The total loss function in this example is:

$$\mathcal{L}(\theta) = w_{\text{goal}}\mathcal{L}_{\text{goal}} + w_{\text{overlap\_goal}}\mathcal{L}_{\text{overlap\_goal}} + w_{\text{vol}}\mathcal{L}_{\text{vol}}, \qquad (11)$$

with $w_{\text{overlap\_goal}} = -15$ and $w_{\text{goal}} = 8$.

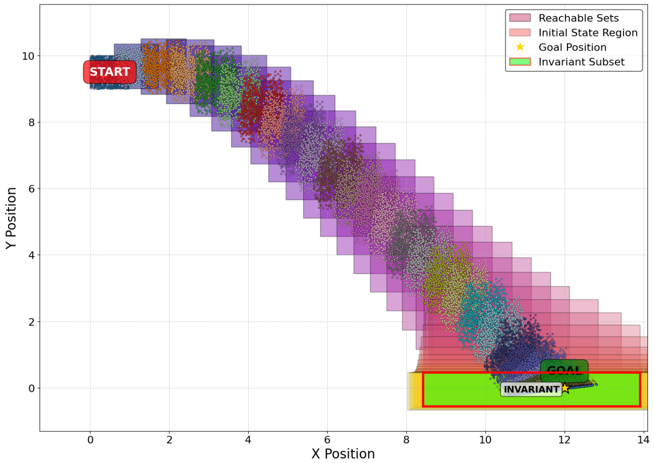


Fig. 4: Effect of Invariance Loss, $\mathcal{L}_{\text{inv}}$. Without considering this loss, the reachable sets often continue to grow or go past the goal (e.g., see Fig. 3). Here, with $w_{\text{inv}} = 100$, the reachable set bounds at the 22nd timestep (bright green set) is a forward invariant set, i.e., the system will never leave this set once it enters. This was confirmed by checking that this set's next reachable set (using CROWN) is a subset.

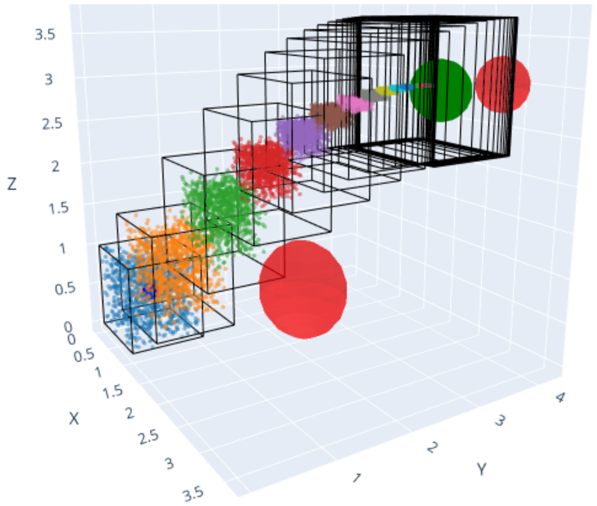


Fig. 5: Obstacle avoidance with 6D quadrotor model (xyz positions shown). The trained policy starts in the set with blue samples and reaches the goal (green) in 20 timesteps. Along the way, both the true system trajectories (samples) and reachable set bounds (calculated with CROWN) avoid the obstacles.

C. Invariance

Fig. 4 shows that including $\mathcal{L}_{\text{inv}}$ from (8) in the loss function can enable finding an invariant set around the goal. In particular, the algorithm noted that $\bar{\mathcal{R}}_{23} \subseteq \bar{\mathcal{R}}_{22}$. This means that there is no state in $\bar{\mathcal{R}}_{22}$ that could lead to a state *not* in $\bar{\mathcal{R}}_{22}$ in one timestep (and thus forever), which makes $\bar{\mathcal{R}}_{22}$ a forward invariant set.

This experiment used $\mathcal{X}_0 = [[0,1],[9,10],[0,\pi/6]]$, obstacles at $(8,8) \pm 0.5$ and $(4,4) \pm 0.5$, goal at $(12,0) \pm 0.5$, $w_{\text{goal}} = 8$, $w_{\text{obstacle\_overlap}} = 20$, $w_{\text{goal\_overlap}} = 15$, $w_{\text{inv}} = 100$, $t_{\text{inv}} = 22$, $T = 40$, and trained with learning rate of 1e-4. The loss plateaued at 1,293 after 16,400 epochs, and the invariant set was $x \in [8.417, 13.911]$ and $y \in [-0.553, 0.458]$.

Recent work also investigated finding invariant sets for systems with learned control policies [41]. While [41] focused on finding the largest possible invariant set for an already trained policy using semidefinite programming (SDP), our approach focuses on training the policy to encourage the existence of an invariant set. Since CROWN is generally faster and scales to larger NNs compared to SDP-based verifiers [12], the proposed approach has a key advantage of being able to bias the training process such that only a linear relaxation-based method was needed to find an invariant set.

D. Scalability: 6D Quadrotor

To demonstrate the scalability of the method, Fig. 5 shows the reachable sets in (x,y,z) position for a 6D quadrotor model. More specifically, this is a model with 3 double integrators that includes drag and coupling between directions,

$$\begin{aligned}
x_{t+1} &= x_t + v_{x,t}\Delta t + (a_{x,t} + c_c v_{y,t} v_{z,t} - c_d v_{x,t}|v_{x,t}|)\frac{\Delta t^2}{2} \\
y_{t+1} &= y_t + v_{y,t}\Delta t + (a_{y,t} + c_c v_{z,t} v_{x,t} - c_d v_{y,t}|v_{y,t}|)\frac{\Delta t^2}{2} \\
z_{t+1} &= z_t + v_{z,t}\Delta t + (a_{z,t} + c_c v_{x,t} v_{y,t} - c_d v_{z,t}|v_{z,t}|)\frac{\Delta t^2}{2} \\
v_{x,t+1} &= v_{x,t} + (a_{x,t} + c_c v_{y,t} v_{z,t} - c_d v_{x,t}|v_{x,t}|)\Delta t \\
v_{y,t+1} &= v_{y,t} + (a_{y,t} + c_c v_{x,t} v_{z,t} - c_d v_{y,t}|v_{y,t}|)\Delta t \qquad (12) \\
v_{z,t+1} &= v_{z,t} + (a_{z,t} + c_c v_{x,t} v_{y,t} - c_d v_{z,t}|v_{z,t}|)\Delta t,
\end{aligned}$$

with $\Delta t = 0.4$, $c_d = 0.01$, $c_c = 0.005$, and let $\mathbf{p} = [x,y,z], \mathbf{v} = [v_x, v_y, v_z], \mathbf{x} = [\mathbf{p}, \mathbf{v}], \mathbf{u} = [a_x, a_y, a_z]$.

Other parameters include initial state set $\mathcal{X}_0 = [[0,1],[0,1],[0,1],[-0.5,0.5],[-0.5,0.5],[-0.5,0.5]]$, goal point $(3,3,3)$, one obstacle at $[2.5, 1.5, 1.0]$ with radius 0.5, another obstacle at $[3.0, 4.0, 3.0]$ with radius 0.3, and $T = 20$. The NN controller with hidden layer sizes $[24, 48, 24]$ was trained for 3,000 epochs to a loss of 2,382 using Adam with learning rate 3e-3.

For this system, the full loss function was,

$$\begin{aligned}
\mathcal{L}_{\text{quad}}(\theta) = & w_{\text{goal}}\mathcal{L}_{\text{goal}} + w_{\text{overlap}}\mathcal{L}_{\text{overlap}} + w_{\text{vel}}\mathcal{L}_{\text{vel}} + w_{\text{vol}}\mathcal{L}_{\text{vol}} + \\
& w_{\text{obs\_entry}}\mathcal{L}_{\text{obs\_entry}} + w_{\text{obs\_prox}}\mathcal{L}_{\text{obs\_prox}}, \qquad (13)
\end{aligned}$$

with weights $w_{\text{goal}} = 50$, $w_{\text{overlap}} = -50$, $w_{\text{vel}} = 0.05$, $w_{\text{vol}} = 40$, $w_{\text{obs\_entry}} = 500$, and $w_{\text{obs\_prox}} = 100$. Along with $\mathcal{L}_{\text{goal}}$ from (6), $\mathcal{L}_{\text{overlap}}$ from (5), and $\mathcal{L}_{\text{vol}}$ from (7), this experiment additionally penalized non-zero velocities:

$$\mathcal{L}_{\text{vel}} = \sum_{t=0}^{T} |\bar{\mathbf{v}}_t| + |\underline{\mathbf{v}}_t|. \qquad (14)$$

The volume loss also included a constant bias of $-\text{vol}(\mathcal{X}_0)$ at each timestep, and was scaled by $\frac{1}{\text{vol}(\mathcal{X}_0)}$, but these constants should be able to be removed.

For the obstacle loss terms, let $\mathbf{n}_{t,j} = \max(\underline{x}_t, \min(\bar{x}_t, \mathbf{o}_j))$ be the nearest point on $\bar{\mathcal{R}}_\sqcup$ to the center of the j-th obstacle, $\mathbf{o}_j$, whose radius is r_j. Then,

$$\mathcal{L}_{\text{obs\_entry}} = \sum_{t=0}^{T} \max(r_j - \|\mathbf{n}_{t,j} - \mathbf{o}_j\|_2, 0)^2 \tag{15}$$

penalizes reachable sets within an obstacle's radius, and

$$\mathcal{L}_{\text{obs\_prox}} = \sum_{t=0}^{T} \sum_{j=0}^{A} \max(r_j + m - \|\mathbf{n}_{t,j} - \mathbf{o}_j\|_2, 0)^2 \tag{16}$$

penalizes reachable sets that get closer than a distance $m = 1$ to each obstacle.

Across the 20 timesteps in the trajectory, none of the reachable sets intersect with the obstacles and the samples move toward the goal region (but we did not optimize for invariance here).

We hypothesize that the training process used here would make relaxed forward and backward reachability approaches less reliant on partitioning schemes, as in [42], which can be difficult to scale to higher dimensional systems. However, we leave that analysis for future work.

V. Conclusion

This paper proposed a new approach for learning NN control policies based on safety specifications. In particular, the approach uses CROWN [1] to calculate reachable sets at each training iteration, then employs loss terms to encourage the system to robustly achieve the safety specs. Numerical experiments on a quadrotor model and unicycle model highlight the ability of this approach to lead to learned control policies that satisfy desired reach-avoid and invariance specifications. Future work will investigate further scalability to larger systems, specifications for more challenging objectives, and integration with imitation learning for safely learning from expert demonstrations.

References

[1] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[2] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon, "Towards fast computation of certified robustness for relu networks," in *International Conference on Machine Learning (ICML)*, 2018, pp. 5276–5285.

[3] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh, "Automatic perturbation analysis for scalable certified robustness and beyond," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 1129–1141, 2020.

[4] A. Raghunathan, J. Steinhardt, and P. S. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[5] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *International Conference on Learning Representations (ICLR)*, 2018.

[6] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić *et al.*, "The marabou framework for verification and analysis of deep neural networks," in *International Conference on Computer-Aided Verification (CAV)*, 2019, pp. 443–452.

[7] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *International Conference on Computer-Aided Verification (CAV)*, 2017, pp. 97–117.

[8] K. Jia and M. Rinard, "Verifying low-dimensional input neural networks via input quantization," in *International Static Analysis Symposium*, 2021, pp. 206–214.

[9] J. A. Vincent and M. Schwager, "Reachable polyhedral marching (RPM): A safety verification algorithm for robotic systems with deep neural network components," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 9029–9035.

[10] C. Sidrane, A. Maleki, A. Irfan, and M. J. Kochenderfer, "Overt: An algorithm for safety verification of neural network control policies for nonlinear systems," *Journal of Machine Learning Research*, vol. 23, no. 117, pp. 1–45, 2022.

[11] S. Chen, V. M. Preciado, and M. Fazlyab, "One-shot reachability analysis of neural network dynamical systems," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 10 546–10 552.

[12] M. Everett, G. Habibi, C. Sun, and J. P. How, "Reachability analysis of neural feedback loops," *IEEE Access*, vol. 9, pp. 163 938–163 953, 2021.

[13] K. D. Julian and M. J. Kochenderfer, "A reachability method for verifying dynamical systems with deep neural network controllers," *arXiv preprint arXiv:1903.00520*, 2019.

[14] H. Hu, M. Fazlyab, M. Morari, and G. J. Pappas, "Reach-SDP: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming," in *IEEE Conference on Decision and Control (CDC)*, 2020, pp. 5929–5934.

[15] Y. Wang, W. Zhou, J. Fan, Z. Wang, J. Li, X. Chen, C. Huang, W. Li, and Q. Zhu, "Polar-express: Efficient and precise formal reachability analysis of neural-network controlled systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 3, pp. 994–1007, 2023.

[16] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: verifying safety properties of hybrid systems with neural network controllers," in *International Conference on Hybrid Systems: Computation and Control*, 2019, pp. 169–178.

[17] S. Dutta, X. Chen, and S. Sankaranarayanan, "Reachability analysis for neural feedback systems using regressive polynomial rule inference," in *International Conference on Hybrid Systems: Computation and Control*, 2019, pp. 157–168.

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

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

[20] W. Xiang, H.-D. Tran, X. Yang, and T. T. Johnson, "Reachable set estimation for neural network control systems: A simulation-guided approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 5, pp. 1821–1830, 2020.

[21] S. Bak and H.-D. Tran, "Neural network compression of ACAS Xu early prototype is unsafe: Closed-loop verification through quantized state backreachability," in *NASA Formal Methods*, 2022, pp. 280–298.

[22] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification," *arXiv preprint arXiv:2103.06624*, 2021.

[23] Y. Zhang and X. Xu, "Safety verification of neural feedback systems based on constrained zonotopes," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 2737–2744.

[24] S. Gowal, K. D. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli, "Scalable verified training for provably robust image classification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4842–4851.

[25] Y.-C. Chang, N. Roohi, and S. Gao, "Neural Lyapunov control," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[26] D. Sun, S. Jha, and C. Fan, "Learning certified control using contraction metric," in *Conference on Robot Learning*. PMLR, 2021, pp. 1519–1539.

[27] M. Han, L. Zhang, J. Wang, and W. Pan, "Actor-critic reinforcement learning for control with stability guarantee," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6217–6224, 2020.

[28] Z. Qin, K. Zhang, Y. Chen, J. Chen, and C. Fan, "Learning safe multi-agent control with decentralized neural barrier certificates," in *International Conference on Learning Representations (ICLR)*, 2020.

[29] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake, "Lyapunov-stable neural-network control," in *Robotics: Science and Systems*, Virtual, Jul. 2021. [Online]. Available: https://arxiv.org/pdf/2109.14152.pdf

[30] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural Lyapunov, barrier, and contraction methods," *arXiv preprint arXiv:2202.11762*, 2022.

[31] T. Badings, W. Koops, S. Junges, and N. Jansen, "Learning-based verification of stochastic dynamical systems with neural network policies," *arXiv preprint arXiv:2406.00826*, 2024.

[32] L. Yang, H. Dai, Z. Shi, C.-J. Hsieh, R. Tedrake, and H. Zhang, "Lyapunov-stable neural control for state and output feedback: A novel formulation," *arXiv preprint arXiv:2404.07956*, 2024.

[33] Y. Wang, C. Huang, Z. Wang, Z. Wang, and Q. Zhu, "Verification in the loop: Correct-by-construction control learning with reach-avoid guarantees," *arXiv preprint arXiv:2106.03245*, 2021.

[34] Y. Wang, S. Zhan, Z. Wang, C. Huang, Z. Wang, Z. Yang, and Q. Zhu, "Joint differentiable optimization and verification for certified reinforcement learning," in *Proceedings of the 2023 ACM/IEEE 14th International Conference on Cyber-Physical Systems (ICCPS)*. ACM, 2023, pp. 132–141. [Online]. Available: https://arxiv.org/abs/2201.12243v1

[35] J. Wu, H. Zhang, and Y. Vorobeychik, "Verified safe reinforcement learning for neural network dynamic models," *arXiv preprint arXiv:2405.15994*, 2024.

[36] R. Bunel, J. Uesato, and L. Berrada, *jax_verify*, 8 2023. [Online]. Available: https://github.com/google-deepmind/jax_verify

[37] Z. Shi, K. Xu, and H. Zhang, *auto_LiRPA*, 3 2025. [Online]. Available: https://github.com/Verified-Intelligence/auto_LiRPA

[38] C. Sidrane and J. Tumova, "Ttt: A temporal refinement heuristic for tenuously tractable discrete time reachability problems," *arXiv preprint arXiv:2407.14394*, 2024.

[39] N. Rober and J. P. How, "Constraint-aware refinement for safety verification of neural feedback loops," *IEEE Control Systems Letters*, 2024.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[41] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," *IEEE Transactions on Automatic Control*, vol. 67, no. 1, pp. 1–15, 2020.

[42] N. Rober, S. M. Katz, C. Sidrane, E. Yel, M. Everett, M. J. Kochenderfer, and J. P. How, "Backward reachability analysis of neural feedback loops: Techniques for linear and nonlinear systems," *IEEE Open Journal of Control Systems*, vol. 2, pp. 108–124, 2023.