Verifiability-Aware Training for Efficient Reachability Analysis of Deep Reinforcement Learning Systems

Qixuan Cao
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, China
51265902038@stu.ecnu.edu.cn

Min Zhang*
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, China
zhangmin@sei.ecnu.edu.cn

***Abstract*—Deep reinforcement learning (DRL) is increasingly used in safety-critical settings, where safety properties need to be formally verified. Reachability analysis can provide sound guarantees, but verifying DRL systems is often expensive due to dual over-approximation of plant dynamics and neural policy outputs. Piece-wise Linear Decision Neural Network (PLDNN) alleviates this difficulty by learning a region-wise linear policy via state abstraction and exporting the closed-loop system as a hybrid automaton amenable to standard reachability tools. Despite this progress, verification can still be slow when the extracted linear control units (LCUs) exhibit (i) action discontinuities across neighboring regions and (ii) high sensitivity within a region, which accelerate reachable-set inflation and increase verification runtime. We propose a lightweight, training-time regularization framework that directly shapes LCUs while keeping the PLDNN abstraction-to-hybrid pipeline unchanged. Specifically, we add a neighbor consistency regularizer to align actions on shared boundaries and a slope penalty to suppress large linear coefficients. Experiments on multiple benchmarks show that our approach reduces verification runtime by up to 37.23% while maintaining comparable control performance to the PLDNN baseline.**

***Index Terms*—deep reinforcement learning verification, reachability analysis, verifiability-aware training**

I. Introduction

Deep reinforcement learning (DRL) has achieved strong performance in a wide range of control problems [1]–[3], and is increasingly explored in safety-critical domains such as robotics and autonomous driving [4], [5]. In these settings, safety cannot rely on testing alone, since rare but dangerous behaviors may never appear in finite simulations [6]. Formal reachability analysis provides stronger guarantees by computing an over-approximation of all states that a closed-loop system can reach [7], [8]. As reachability analysis is increasingly used not only offline but also in runtime assurance with tight time budgets [9]–[11], the need for efficient reachability analysis becomes even more pressing. Unfortunately, reachability analysis for DRL systems is often computationally expensive [8], [12], [13]. This motivates verification methods that are both sound and efficient for DRL systems.

*Corresponding author.

To obtain *sound* reachability results for DRL systems, many methods treat the deep neural network (DNN) controller as a white box and over-approximate its output over a *set* of states [8], [12]. However, this often leads to a *dual over-approximation*: over-approximating both the dynamics evolution and the neural controller output. As a result, the reachable sets can quickly become overly conservative and the computational cost can grow rapidly, which severely limits practical reachability analysis for DRL systems.

A representative alternative is PLDNN [14], which alleviates the dual over-approximation issue by training the policy with state abstraction. Specifically, the learned PLDNN can be extracted as a set of region-wise linear control units (LCUs), one per abstract state. This structure enables exporting the closed-loop system as a hybrid automaton, which can then be verified using existing hybrid-system reachability tools such as Flow* [7]. This abstraction-based method preserves near-optimal control performance [15], [16], while making verification much more tractable than prior white-box approaches which verify a DNN layer by layer [17], suggesting a promising direction for sound DRL verification.

However, even with PLDNN, we find that verification runtime can still be further improved. The remaining bottlenecks largely come from two structural properties of the extracted LCUs that directly interact with reachability computation: **(i) Between-region discontinuity:** When the reachable state set intersects a region boundary, the hybrid encoding must conservatively account for multiple possible LCUs at the next control update. If neighboring regions output noticeably different actions near the shared boundary, then even a small uncertainty around the boundary can induce a large action range after over-approximation. In the hybrid automaton, this manifests as larger (and more heterogeneous) discrete control updates, which often cause abrupt expansion of the reachable-set over-approximation and trigger more splitting, increasing runtime. **(ii) Inside-region sensitivity:** Within one abstract state, an LCU is an affine map $a = b + c^\top s$, where a is the action and s is the actual state. If the slope vector c is large, then a small uncertainty in the state set is amplified into a

much larger uncertainty in the control input. Since the control input affects the dynamics directly, this amplification makes the flowpipes blow up faster during continuous propagation, again increasing splitting and runtime. These issues are not explicitly controlled by the original PLDNN training objective, which mainly focuses on maximizing reward.

To address these bottlenecks, we propose a *verification-friendly* training objective for PLDNN. Our key idea is to keep the same overall pipeline, but modify the policy training objective to shape the extracted LCUs. Concretely, we add two lightweight terms to the policy loss. First, we introduce a **neighbor consistency regularizer** that encourages adjacent LCUs to agree on shared boundaries, so that the control law changes more smoothly across regions. Second, we add a **slope penalty** that discourages overly large linear coefficients (excluding the bias term), which limits the sensitivity of the control input to state uncertainty and helps prevent fast flowpipe growth during reachability computation. In implementation, these two regularizers are added to the original policy training objective with tunable weights, while the rest of the PLDNN training and the downstream verification pipeline remain unchanged. In our experiments, these structure-aware regularizers improve verification efficiency while maintaining comparable control performance.

Our contributions are summarized as follows:

1) We identify two practical sources of verification difficulty for PLDNN controllers: boundary discontinuities between neighboring regions and high sensitivity within a region.
2) We propose two lightweight regularizers that directly address these issues, making the extracted LCUs more verification-friendly. The regularizers are integrated into the training objective without changing the existing verification pipeline.
3) We evaluate the proposed method on multiple benchmarks. The results show that our method reduces verification runtime by up to 37.23% while maintaining control performance comparable to PLDNN.

The remainder of the paper is organized as follows. Section III introduces the necessary preliminaries used in this paper while Section IV presents our proposed method. Section V demonstrates the experimental results, and Section VI concludes the paper.

II. Related Work

Reachability analysis for DRL systems. Reachability analysis for DRL systems and more generally neural network control systems (NNCSs) typically over-approximates the neural network controller output over a *set* of states and propagates it together with the plant dynamics to compute a sound reachable set. POLAR [8] improves Taylor-model-based reachability for NNCSs using polynomial arithmetic with Bernstein–Bézier forms and symbolic remainders. Verisig 2.0 [13] targets `tanh`/`sigmoid` networks and scales closed-loop reachability via preconditioning and shrink-wrapping. ReachNN* [18] abstracts neural controllers with Bernstein-polynomial approximations and provable error bounds, combined with sampling-based tightening. Compared with these methods, PLDNN [14] takes a structurally different and promising route: it trains a piece-wise linear controller via state abstraction and exports the closed-loop system as a hybrid automaton, enabling reachability analysis with standard hybrid-system verification tools such as Flow* [7]. In this paper, we further improve the verification efficiency of the exported hybrid model, while keeping the overall PLDNN pipeline unchanged.

Regularization in training. Adding regularization terms to the training objective [19] is a common way to encourage desired properties that are not enforced by the original goal (e.g., accuracy or reward). For example, Sharpness-Aware Minimization (SAM) improves generalization by optimizing for parameters that have low loss in a local neighborhood, rather than only at a single point [20]. R-Drop adds a simple consistency regularizer on top of dropout by penalizing the mismatch between two stochastic forward passes of the same input [21]. Relatedly, Lipschitz-based regularization has been used to suppress overly sensitive functions by directly penalizing an upper bound on the Lipschitz constant [22]. Motivated by this general idea, we address the verification bottlenecks of PLDNN by regularizing the training objective. Instead of reusing generic regularizers, we introduce two PLDNN-specific terms tailored to the two bottlenecks.

III. Preliminaries

A. Deep Reinforcement Learning Systems

Formally, a DRL system [23] can be modeled as a tuple $M = (S, S_0, A, \pi, f, R)$, where $S \subseteq \mathbb{R}^n$ is the state space, $S_0 \subseteq S$ is the set of initial states, A is the action space, $\pi : S \to A$ is the NN-based policy mapping states to actions, $f : S \times A \to \dot{S}$ is the system dynamics, and $R : S \times A \to \mathbb{R}$ is the reward function. The DRL system proceeds in discrete time steps $t \in \mathbb{N}_0$, each of which spans a fixed duration δ. At the start of each time step, the agent observes the current state $s_t \in S$ and applies the neural policy to select an action $a_t = \pi(s_t)$. This action is then held constant over the next interval of length δ, during which the state evolves according to the continuous-time dynamics $\dot{s}(\tau) = f(s(\tau), a_t)$ for $\tau \in [0, \delta)$. Consequently, the next state can be expressed as $s_{t+1} = s_t + \int_0^\delta f(s(\tau), a_t)\, d\tau$. The system also returns a scalar reward $r_{t+1} = R(s_t, a_t)$, which serves as feedback for policy learning.

Learning objective. The goal of DRL is to maximize the expected (discounted) cumulative reward. In this paper, we follow the standard actor–critic setting used in PLDNN (e.g., DDPG [1]). The critic (action-value) function is denoted by $Q : S \times A \to \mathbb{R}$, which estimates the expected return of taking action a at state s under the current policy. Accordingly, the actor is trained by minimizing the loss

$$\mathcal{L}_{\text{actor}} \;=\; -\,\mathbb{E}[\,Q(s, \pi(s))\,]\,. \tag{1}$$

B. Reachability Problem of DRL Systems

The reachability problem of a DRL system is to compute an (over-approximated) set of states reachable from an

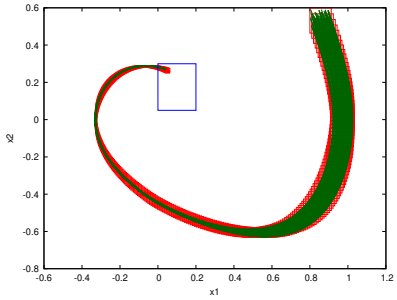

Fig. 1: Reachability Analysis.

initial set S_0 under the closed-loop dynamics induced by the policy π over a given horizon T. Given a DRL system $M = (S, S_0, A, \pi, f, R)$, starting from an initial state s_0, the state at time $t = n\delta + t'$ for some integer $n \geq 0$ and $0 \leq t' < \delta$ is defined as:

$$\varphi_f(s_0, \pi, t) = s_n + \int_0^{t'} f(s(\tau), \pi(s_n))\, d\tau, \tag{2}$$

where $s_{i+1} = s_i + \int_0^{\delta} f(s(\tau), \pi(s_i))d\tau$ for all $i \in \{0, \ldots, n-1\}$. Given a set of initial states S_0, the reachable set of states at time T and over $[0, T]$ are denoted by $Reach_T(S_0)$ and $Reach_{[0,T]}(S_0)$, respectively, where:

$$Reach_T(S_0) = \{\varphi_f(s_0, \pi, T) \mid s_0 \in S_0\} \tag{3}$$

$$Reach_{[0,T]}(S_0) = \{\varphi_f(s_0, \pi, t) \mid t \in [0, T], s_0 \in S_0\} \tag{4}$$

In general, computing $Reach_T(S_0)$ (or $Reach_{[0,T]}(S_0)$) exactly is intractable for nonlinear dynamics and neural-network policies. Reachability tools therefore compute sound over-approximations of these sets, as illustrated in Fig. 1, where the blue box is the goal region, the green lines are simulation trajectories and the red boxes are over-approximation sets.

C. Piece-wise Linear Decision Neural Networks

PLDNN [14] is trained based on state abstraction. The actual system states are abstracted as intervals. Given an n-dimensional system with a bounded continuous state space $S = \prod_{i=1}^{n}[L_i, U_i]$ where $L_i, U_i \in \mathbb{R}$, based on the abstraction granularity $\gamma = (d_1, d_2, ..., d_n)$, the i-th dimension will be divided evenly into $(U_i - L_i)/d_i$ intervals. In this way, each abstract state can be represented as a $2n$-dimensional vector $\alpha = (l_1, u_1, \ldots, l_n, u_n)$, where $[l_i, u_i]$ specifies the interval of this abstract state in the i-th dimension. The abstraction mapping from a concrete state to its corresponding abstract state is denoted by $\textsc{Abstract}(\cdot)$: for any $s = (s_1, \ldots, s_n) \in S$, $\alpha \leftarrow \textsc{Abstract}(s)$ returns the unique abstract state α such that $s_i \in [l_i, u_i]$ for all $i \in \{1, \ldots, n\}$.

By adding an abstraction layer between the input layer and the first hidden layer, PLDNN converts an actual system state into its corresponding abstract state. For all the states that correspond to the same abstract state, PLDNN gives the same output, which is a set of coefficients parameterizing a LCU for that abstract state. Concretely, for an n-dimensional state $s = (s_1, \ldots, s_n)$, PLDNN outputs an $(n+1)$-dimensional coefficient vector $\theta = (b, c_1, c_2, \ldots, c_n)$, and the action is computed as

$$a = b + \sum_{j=1}^{n} c_j s_j \;=\; \theta^{\top}[1; s]. \tag{5}$$

For multi-dimensional actions $a \in \mathbb{R}^m$, the network outputs m such coefficient vectors, one for each action dimension.

For each abstract state, we pick an arbitrary concrete state s and query the PLDNN to obtain its control coefficient vector, which uniquely determines the corresponding LCU. All extracted LCUs together form a policy that is behaviorally equivalent to the original PLDNN. Replacing the neural controller with these LCUs yields a verifiable hybrid automaton of the closed-loop system, which can be efficiently analyzed by tools such as Flow* [7].

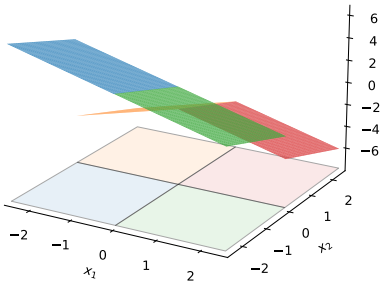

Fig. 2: LCUs extracted from PLDNN.

IV. Method

A. Method Overview

We improve PLDNN within the same abstraction-to-hybrid verification framework, with the goal of making the learned controller *more verification-friendly*. Fig. 2 illustrates a set of LCUs extracted from a trained PLDNN, where x_1 and x_2 are the two state variables and the vertical axis represents the control action $\pi(x_1, x_2)$. This visualization provides an intuitive explanation of the key verification bottlenecks of the exported hybrid model. Specifically, the bottlenecks mainly come from two structural issues of the extracted LCUs: (i) *between-region discontinuity*: neighboring regions may output noticeably different actions near shared boundaries, which enlarges the discrete control-update uncertainty in the hybrid encoding; and (ii) *inside-region sensitivity*: an LCU with large linear coefficients amplifies a small state-set uncertainty into a much larger action uncertainty, causing faster flowpipe inflation during continuous propagation.

To mitigate both issues at training time, we keep the PLDNN pipeline unchanged, but modify the *policy training objective* to directly shape the extracted LCUs. Concretely, we introduce two lightweight regularizers: (1) a **neighbor**

boundary-consistency loss that aligns the actions produced by adjacent LCUs on their shared boundary to reduce boundary-induced nondeterminism (Section IV-B); and (2) a **slope penalty** that suppresses overly large linear coefficients inside each LCU (excluding the bias term) to limit uncertainty amplification during continuous propagation (Section IV-C). We combine these terms with the original policy loss and apply a warm-up schedule so that the regularization gradually takes effect during early exploration. The resulting training objective and procedure are summarized in Section IV-D. Finally, after training we export the learned LCUs into a hybrid automaton following PLDNN, enabling verification by existing standard reachability tools (Section IV-E).

B. Neighbor Boundary Consistency Regularization

A major source of conservatism in verifying piece-wise linear controllers arises from action discontinuities across region boundaries. In the PLDNN framework, each abstract state corresponds to a unique LCU. When the reachable state set intersects a boundary between abstract states, the hybrid encoding must conservatively account for multiple possible LCUs at the next control update. If neighboring LCUs output different actions near the boundary, then even a small uncertainty around it can lead to a large action range after over-approximation, which typically triggers more splitting and increases verification runtime.

To explicitly reduce such boundary-induced conservatism during training, we enforce *boundary consistency* between neighboring abstract states by sampling points on their shared boundary and penalizing action mismatches. Let $\alpha = (l_1, u_1, \ldots, l_n, u_n)$ denote an abstract state. Its neighbor set $N(\alpha)$, consisting of abstract states that share a boundary (i.e., a common $(n-1)$-dimensional face) with α, is defined as

$$N(\alpha) = \Big\{ \alpha' = (l'_1, u'_1, \ldots, l'_n, u'_n) \;\Big|\; \exists\, k \in \{1, \ldots, n\} : (l'_k, u'_k) = (u_k, u_k + d_k) \text{ or } (l'_k, u'_k) = (l_k - d_k, l_k), \text{ and } \forall i \neq k,\ (l'_i, u'_i) = (l_i, u_i) \Big\}, \quad (6)$$

where d_k is the abstraction granularity on the k-th dimension. For each training sample whose state lies in α, we randomly select a neighbor $\alpha' \in N(\alpha)$ and sample a boundary point s_b on the shared boundary between α and α'. Let k be the (unique) dimension along which α and α' touch. We sample s_b by fixing its k-th coordinate to the boundary value and sampling the remaining coordinates uniformly:

$$s_{b,k} = \begin{cases} u_k, & \text{if } u_k = l'_k, \\ l_k, & \text{if } l_k = u'_k, \end{cases} \quad s_{b,i} \sim \mathrm{Unif}([l_i, u_i]) \quad \forall i \neq k, \quad (7)$$

where $\mathrm{Unif}([l_i, u_i])$ denotes the uniform distribution over the interval $[l_i, u_i]$. We encourage the two neighboring LCUs to output consistent actions at the shared boundary by minimizing

$$\mathcal{L}_{bc} \;=\; \mathbb{E}\Big[\, \|\pi_\alpha(s_b) - \pi_{\alpha'}(s_b)\|_2^2 \Big], \quad (8)$$

where π_α and $\pi_{\alpha'}$ are the LCUs corresponding to α and α', respectively, and the expectation is taken over training samples, the random choice of $\alpha' \in N(\alpha)$, and the sampled boundary point s_b. Minimizing $\mathcal{L}_{bc}$ reduces the mismatch of discrete control updates when the reachable set crosses region boundaries, and thus helps mitigate reachable-set expansion caused by boundary-induced nondeterminism.

C. Slope Regularization

The boundary-consistency regularizer in Section IV-B addresses discontinuities across regions. We further regularize the *within-region* sensitivity of each LCU, since overly steep LCUs can still cause fast flowpipe growth and slow verification. This inside-region sensitivity is critical for reachability: during flowpipe propagation, the verifier maintains an over-approximated set of possible states. If a small state-set uncertainty already induces a large action uncertainty, the dynamics are effectively driven by a wide range of control inputs, which causes faster reachable-set inflation and often triggers additional splitting and longer runtime.

This issue can be seen directly from the affine form of an LCU. For an abstract state α, the LCU can be written as

$$\pi_\alpha(s) \;=\; b_\alpha + \sum_{j=1}^{n} c_{\alpha,j}\, s_j \;=\; b_\alpha + c_{\alpha,1:n}^\top s. \quad (9)$$

For any two concrete states s, s' inside the same abstract state α, the action variation satisfies

$$\big|\pi_\alpha(s) - \pi_\alpha(s')\big| = \big|c_{\alpha,1:n}^\top (s - s')\big| \le \big\|c_{\alpha,1:n}\big\|_2 \cdot \big\|s - s'\big\|_2, \quad (10)$$

which indicates that $\|c_{\alpha,1:n}\|_2$ acts as a local sensitivity (a Lipschitz-like bound) of the control law within α. Therefore, reducing $\|c_{\alpha,1:n}\|_2$ directly limits how much state-set uncertainty can be amplified into action uncertainty, helping slow down flowpipe growth in reachability computation.

Motivated by this, we introduce a slope penalty to suppress overly large linear coefficients within each abstract state:

$$\mathcal{L}_{sl} \;=\; \mathbb{E}\Big[\, \big\|c_{\alpha,1:n}\big\|_2^2 \Big], \quad (11)$$

where the expectation is taken over training samples (each belonging to an abstract state α). We do not penalize the bias term b_α because it mainly shifts the action by a constant and does not amplify state uncertainty like the slope terms in Eq. (10). For multi-dimensional actions, we apply the same penalty to each action dimension and sum them up. In implementation, $\mathcal{L}_{sl}$ is easy to compute: PLDNN directly outputs the LCU coefficients for each abstract state, and we simply average $\|c_{\alpha,1:n}\|_2^2$ over the minibatch. This term can be added to the actor loss with negligible overhead, while discouraging steep LCUs that are unfavorable for verification.

D. Training Objective

We adopt the same actor–critic training framework as PLDNN (e.g., DDPG [1]) and keep the overall pipeline unchanged. The critic Q is still updated with the standard temporal-difference (TD) regression loss. Our method only modifies the actor update by augmenting the original actor

loss in Eq. (1) with additional regularization terms to learn verification-friendly LCUs.

Warm-up of regularization. Applying full regularization too early may hinder exploration and slow learning. To make the regularization effect gradually take place, we use a simple linear warm-up schedule. Let t denote the number of actor update steps and T_w be the warm-up length, and define the warm-up weight as

$$w = \min\left(1, \frac{t}{T_w}\right). \tag{12}$$

Thus w starts from 0 and increases to 1, after which the full regularization strength is applied.

With this warm-up weight, we augment the actor loss using the two regularizers in Sections IV-B and IV-C:

$$\mathcal{L}_{\text{actor}}^{\text{reg}} = -\mathbb{E}\big[Q(s, \pi(s))\big] + (w\lambda_{bc})\,\mathcal{L}_{bc} + (w\lambda_{sl})\,\mathcal{L}_{sl}, \tag{13}$$

where λ_{bc} (Eq. (8)) and λ_{sl} (Eq. (11)) are hyperparameters that control the strengths of boundary consistency and slope regularization, respectively.

In implementation, all expectations are estimated by mini-batch averages. For each sampled state s in a minibatch, we first compute its abstract state $\alpha \leftarrow \textsc{Abstract}(s)$. We then (i) sample a neighbor $\alpha' \in N(\alpha)$ and a boundary point s_b to evaluate the boundary-consistency term $\|\pi_\alpha(s_b) - \pi_{\alpha'}(s_b)\|_2^2$, and (ii) extract the LCU coefficients produced by PLDNN for α to evaluate the slope term $\|c_{\alpha,1:n}\|_2^2$. Both $\mathcal{L}_{bc}$ and $\mathcal{L}_{sl}$ are differentiable with respect to the actor parameters, since $\pi_\alpha(\cdot)$ and $(b_\alpha, c_{\alpha,1:n})$ are outputs of the actor network. Therefore, the actor can be updated by standard backpropagation on Eq. (13), without introducing any extra optimization loop or modifying the downstream verification procedure. The full training procedure is summarized in Algorithm 1.

E. Hybrid Automaton Construction for Verification

After training, we extract one LCU for each abstract state α. Replacing the original DNN controller with this equivalent set of LCUs allows the entire DRL system to be represented as a hybrid automaton. This step follows the same abstraction-to-hybrid procedure as PLDNN [14].

The hybrid automaton models the closed-loop system with plant dynamics $\dot{s} = f(s, a)$, where the control input a is held constant during each time step. To model periodic controller updates with sampling period δ, we introduce a clock variable τ with $\dot{\tau} = 1$ and invariant $\tau \le \delta$. When $\tau = \delta$, a discrete transition is triggered: the clock is reset to $\tau := 0$, and the control input is updated using the LCU associated with the current abstract state. Specifically, if the actual current state s lies in abstract state α, we apply the reset $a := \pi_\alpha(s),\ \tau := 0$. The transition guards encode $s \in \alpha$ (i.e., $l_i \le s_i \le u_i$ for all i), thereby selecting the corresponding LCU at each update. As a result, the neural-network controller is replaced by a finite set of LCUs, yielding a verifiable hybrid automaton. We then use reachability tools such as Flow* [7] to compute sound over-approximations of the reachable set and verify the desired safety property.

Algorithm 1: Training with Boundary Consistency and Slope Regularization (with Warm-up)

Require: Weights $\lambda_{bc}, \lambda_{sl}$; warm-up steps T_w
Initialize actor π, critic Q and replay buffer $\mathcal{D}$
$t \leftarrow 0$
while not converged **do**
  Collect transitions and store them in $\mathcal{D}$
  Sample a minibatch $\mathcal{B} = \{(s, a, r, s')\}$ from $\mathcal{D}$
  Update critic Q using the standard TD loss
  $t \leftarrow t + 1$
  $w \leftarrow \min\left(1, \frac{t}{T_w}\right)$
  for each (s, a, r, s') in $\mathcal{B}$ **do**
    Compute abstract state $\alpha \leftarrow \textsc{Abstract}(s)$
    Extract the slope vector $c_{\alpha,1:n}$ of π_α
    Sample a neighbor $\alpha' \in N(\alpha)$ and a boundary point s_b on the shared boundary
  end for
  $\mathcal{L}_{bc} \leftarrow \mathbb{E}_{\mathcal{B}}\big[\|\pi_\alpha(s_b) - \pi_{\alpha'}(s_b)\|_2^2\big]$
  $\mathcal{L}_{sl} \leftarrow \mathbb{E}_{\mathcal{B}}\big[\|c_{\alpha,1:n}\|_2^2\big]$
  Update actor by minimizing

$$\mathcal{L}_{\text{actor}}^{\text{reg}} = -\mathbb{E}_{\mathcal{B}}\big[Q(s, \pi(s))\big] + w\lambda_{bc}\mathcal{L}_{bc} + w\lambda_{sl}\mathcal{L}_{sl}.$$

end while

V. Experiments

A. Experiment Setup

We first summarize the experimental settings. Our experiments are organized around four research questions: **RQ1**: Does our method improve verification efficiency compared with PLDNN? **RQ2**: Does our method preserve control performance while improving verification efficiency? **RQ3**: How do the two regularizers contribute individually and jointly? (Ablation study) **RQ4**: How do verification time and reward change as the two regularization weights λ_{bc} and λ_{sl} vary? (Hyperparameter study)

Unless otherwise specified, we use $\lambda_{bc} = \lambda_{sl} = 0.1$ in all experiments. For the ablation study (RQ3), when a regularizer is enabled, its corresponding weight is set to 0.1. The only exception is the hyperparameter study in RQ4, where λ_{bc} and λ_{sl} are varied explicitly.

Benchmarks. We evaluate our method on six standard continuous-control benchmarks used in PLDNN from [13] and [24]: B1–B5 and CartPole. For all settings, policies are trained using DDPG [1]. B1–B5 cover different nonlinear dynamics and verification difficulties, while CartPole provides a higher-level control task with different state dimensions. For a fair comparison, we use the same state abstraction, hybrid-automaton export and verification horizon as PLDNN, so the only difference comes from the learned LCUs.

Metrics. We report two main metrics. **Verification time** measures the wall-clock runtime (in seconds) needed to complete reachability verification on the exported hybrid automaton. For verification, we use Flow* [7] with an identical configuration

TABLE I: Verification time in seconds and runtime reduction relative to PLDNN.

Task	Dim	Network	PLDNN	Our Method	Reduction (%)
B1	2	$\text{Tanh}_{2\times20}$	2.182	**1.943**	10.95
		$\text{Tanh}_{3\times100}$	2.154	**1.352**	37.23
		$\text{ReLU}_{2\times20}$	1.998	**1.820**	8.91
		$\text{ReLU}_{3\times100}$	2.442	**1.645**	32.64
B2	2	$\text{Tanh}_{2\times20}$	0.610	**0.544**	10.82
		$\text{Tanh}_{3\times100}$	0.573	**0.545**	4.89
		$\text{ReLU}_{2\times20}$	0.613	**0.540**	11.91
		$\text{ReLU}_{3\times100}$	0.575	**0.544**	5.39
B3	2	$\text{Tanh}_{2\times20}$	2.529	**2.469**	2.37
		$\text{Tanh}_{3\times100}$	3.353	**2.432**	27.47
		$\text{ReLU}_{2\times20}$	2.885	**2.544**	11.82
		$\text{ReLU}_{3\times100}$	2.754	**2.283**	17.10
B4	3	$\text{Tanh}_{2\times20}$	1.334	**1.110**	16.79
		$\text{Tanh}_{3\times100}$	1.340	**1.114**	16.87
		$\text{ReLU}_{2\times20}$	1.331	**1.111**	16.53
		$\text{ReLU}_{3\times100}$	1.349	**1.112**	17.57
B5	3	$\text{Tanh}_{3\times100}$	3.034	**2.214**	27.03
		$\text{Tanh}_{4\times200}$	3.083	**2.215**	28.15
		$\text{ReLU}_{3\times100}$	3.082	**2.208**	28.36
		$\text{ReLU}_{4\times200}$	3.079	**2.211**	28.19
CartPole	4	$\text{Tanh}_{3\times64}$	105.887	**100.916**	4.69

for PLDNN and our method on each task, including the integration step size, Taylor model orders, and remainder estimation, unless otherwise stated. **Control performance** is measured by the **average episodic reward** of the learned policy in the environment. For each policy and each benchmark, we evaluate the policy for 1000 episodes and report the mean reward and standard deviation.

Platform. All experiments were conducted on a computer equipped with an Intel Core i9-13900K CPU and 64GB RAM, running Ubuntu 20.04.6 LTS.

B. Experiment Results

RQ1: Does our method improve verification efficiency compared with PLDNN?

Table I reports the verification time of PLDNN and our method within each task, together with the relative reduction achieved by our method. In Table I, $\text{Tanh}_{L\times H}$ and $\text{ReLU}_{L\times H}$ denote fully connected MLPs with L hidden layers of width H; the hidden-layer activations are Tanh or ReLU, respectively, while the output layer always uses Tanh.

Overall, our method *consistently* reduces verification runtime in all settings, indicating that shaping LCUs during training makes the exported hybrid models more amenable to reachability analysis. As a representative example, one of the largest reductions is on B1 with $\text{Tanh}_{3\times100}$, where our method cuts the runtime from 2.154s to 1.352s (a 37.23% decrease).

RQ2: Does our method preserve control performance while improving verification efficiency?

Table II reports the average episodic reward (mean $\pm$ std) for three controllers: Standard-RL, PLDNN, and our method. For each task–network setting, we evaluate each controller over 1000 episodes and compute the mean and standard deviation of the episodic returns. Here *Standard-RL* denotes a conventional RL policy trained directly on the original (continuous) state space without state abstraction, and serves as a reference for near-optimal control performance. PLDNN argues that abstraction-based training can still achieve near-optimal performance relative to this Standard-RL baseline [14]; our method preserves this claim after introducing verification-oriented regularization, as its rewards remain comparable to PLDNN and close to the Standard-RL reference.

TABLE II: Control performance (average reward, mean $\pm$ std).

Task	Network	Standard-RL	PLDNN	Our Method
B1	$\text{Tanh}_{2\times20}$	52.93 ± 0.916	56.67 ± 0.823	$\mathbf{56.79 \pm 0.788}$
	$\text{Tanh}_{3\times100}$	53.51 ± 0.829	$\mathbf{56.75 \pm 0.711}$	56.41 ± 0.769
	$\text{ReLU}_{2\times20}$	53.10 ± 0.918	$\mathbf{57.40 \pm 0.700}$	56.86 ± 0.737
	$\text{ReLU}_{3\times100}$	52.85 ± 0.959	$\mathbf{57.29 \pm 0.782}$	57.14 ± 0.756
B2	$\text{Tanh}_{2\times20}$	91.54 ± 0.597	$\mathbf{94.65 \pm 0.194}$	94.60 ± 0.199
	$\text{Tanh}_{3\times100}$	91.53 ± 0.574	$\mathbf{95.45 \pm 0.200}$	94.67 ± 0.180
	$\text{ReLU}_{2\times20}$	91.38 ± 0.566	$\mathbf{95.05 \pm 0.183}$	94.65 ± 0.175
	$\text{ReLU}_{3\times100}$	91.41 ± 0.613	$\mathbf{95.60 \pm 0.207}$	94.65 ± 0.182
B3	$\text{Tanh}_{2\times20}$	$\mathbf{192.57 \pm 0.243}$	191.94 ± 0.229	191.99 ± 0.231
	$\text{Tanh}_{3\times100}$	$\mathbf{192.21 \pm 0.231}$	191.31 ± 0.276	191.98 ± 0.222
	$\text{ReLU}_{2\times20}$	192.63 ± 0.226	$\mathbf{193.06 \pm 0.204}$	192.16 ± 0.223
	$\text{ReLU}_{3\times100}$	192.50 ± 0.241	$\mathbf{192.93 \pm 0.200}$	191.07 ± 0.332
B4	$\text{Tanh}_{2\times20}$	$\mathbf{498.19 \pm 0.093}$	498.16 ± 0.102	498.17 ± 0.103
	$\text{Tanh}_{3\times100}$	$\mathbf{498.76 \pm 0.038}$	498.16 ± 0.106	498.17 ± 0.099
	$\text{ReLU}_{2\times20}$	$\mathbf{498.28 \pm 0.063}$	498.16 ± 0.101	498.17 ± 0.101
	$\text{ReLU}_{3\times100}$	$\mathbf{498.76 \pm 0.040}$	498.17 ± 0.101	498.17 ± 0.102

TABLE III: Ablation study on verification time in seconds for the two regularizers.

Task	Network	PLDNN	PLDNN+$\mathcal{L}_{bc}$	PLDNN+$\mathcal{L}_{sl}$	PLDNN+$\mathcal{L}_{bc}$+$\mathcal{L}_{sl}$
B1	$\text{Tanh}_{3\times100}$	2.154	2.101	1.488	**1.352**
	$\text{ReLU}_{3\times100}$	2.442	2.103	2.351	**1.645**
B3	$\text{Tanh}_{3\times100}$	3.353	2.598	2.564	**2.432**
	$\text{ReLU}_{3\times100}$	2.754	2.562	2.595	**2.283**

Overall, our method preserves control performance across all tasks and network architectures, indicating that the added regularizers do not noticeably degrade the learned policy. Even in the worst case in Table II (B3 with $\text{ReLU}_{3\times100}$), the reward drop relative to PLDNN is small compared to the task scale, and our method remains in the same performance regime.

RQ3: How do the two regularizers contribute individually and jointly? (Ablation Study)

We conduct an ablation study by enabling each regularizer separately and together. Table III reports the verification time of the exported hybrid automata.

In general, both $\mathcal{L}_{bc}$ and $\mathcal{L}_{sl}$ contribute to faster verification, and their effects are complementary. Using either regularizer alone already reduces verification time compared with the baseline, while combining them (baseline+$\mathcal{L}_{bc}$+$\mathcal{L}_{sl}$) consistently achieves the lowest time across all tested settings. This indicates that boundary consistency and slope regularization address different sources of verification difficulty, and applying them together provides the most stable improvement.

RQ4: How do verification time and reward change when we vary the two regularization weights λ_{bc} and λ_{sl}? (Hyperparameter Study)

TABLE IV: Hyperparameter study: reward (mean $\pm$ std) and verification time (s) under different λ_{bc} and λ_{sl}.

Task	λ_{bc}	λ_{sl}	Reward	Verification Time (s)
B1	0	0	56.71 ± 0.696	2.154
	0.01	0.01	56.60 ± 0.842	1.717
	0.1	0.01	57.05 ± 0.761	1.962
	0.01	0.1	56.78 ± 0.668	1.518
	0.1	0.1	56.41 ± 0.769	**1.352**
	1	0.1	55.66 ± 0.694	1.700
	0.1	1	54.51 ± 0.784	1.367
	1	1	55.14 ± 0.736	**1.352**

We vary the two regularization weights λ_{bc} and λ_{sl} and report both verification time and average reward in Table IV. Overall, moderate regularization provides the best trade-off: increasing either weight from zero generally reduces verification time, while overly large weights can noticeably lower reward. In our sweep, the best overall setting is $\lambda_{bc} = \lambda_{sl} = 0.1$, which achieves the fastest verification while keeping the reward close to the unregularized baseline; we therefore adopt this configuration in the main experiments reported above. By contrast, pushing one of the weights to 1 tends to reduce reward and does not further improve verification time in a consistent way. These results suggest that the method is not overly sensitive to the exact choice of weights, but selecting λ_{bc} and λ_{sl} within a moderate range is important for obtaining verification gains without sacrificing control performance.

VI. Conclusion

In this paper, we present a verifiability-aware training objective to further improve the practicality of PLDNN-style DRL verification. By analyzing how region-wise linear control units (LCUs) interact with hybrid reachability, we identify two verification bottlenecks: between-region discontinuity and inside-region sensitivity, and introduce two targeted regularizers to mitigate them during training. Experiments show that our method consistently reduces verification runtime (up to 37.23%) while preserving control performance comparable to the PLDNN baseline. Future work will further explore verification-friendly training objectives that are more tightly aligned with reachability computation.

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2016.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*. Pmlr, 2018, pp. 1861–1870.

[3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[4] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[5] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[6] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," *Advances in Neural Information Processing systems (NeurIPS)*, vol. 31, 2018.

[7] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *International Conference on Computer Aided Verification (CAV)*. Springer, 2013, pp. 258–263.

[8] C. Huang, J. Fan, X. Chen, W. Li, and Q. Zhu, "Polar: A polynomial arithmetic framework for verifying neural-network controlled systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2022, pp. 414–430.

[9] C. Pek, S. Manzinger, M. Koschi, and M. Althoff, "Using online verification to prevent autonomous vehicles from causing accidents," *Nature Machine Intelligence*, vol. 2, no. 9, pp. 518–528, 2020.

[10] P. Musau, N. Hamilton, D. M. Lopez, P. Robinette, and T. T. Johnson, "On using real-time reachability for the safety assurance of machine learning controllers," in *2022 IEEE International Conference on Assured Autonomy (ICAA)*. IEEE, 2022, pp. 1–10.

[11] F. Yang, S. S. Zhan, Y. Wang, C. Huang, and Q. Zhu, "Case study: runtime safety verification of neural network controlled system," in *International Conference on Runtime Verification*. Springer, 2024, pp. 205–217.

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

[13] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. Pappas, and I. Lee, "Verisig 2.0: Verification of neural network controllers using taylor model preconditioning," in *International Conference on Computer Aided Verification (CAV)*. Springer, 2021, pp. 249–262.

[14] J. Tian, D. Zhi, S. Liu, P. Wang, C. Chen, and M. Zhang, "Boosting verification of deep reinforcement learning via piece-wise linear decision neural networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, pp. 10 022–10 037, 2023.

[15] D. Abel, D. Hershkowitz, and M. Littman, "Near optimal behavior via approximate state abstraction," in *International Conference on Machine Learning (ICML)*. PMLR, 2016, pp. 2915–2923.

[16] P. Jin, J. Tian, D. Zhi, X. Wen, and M. Zhang, "Trainify: A cegar-driven training and verification framework for safe deep reinforcement learning," in *International Conference on Computer Aided Verification (CAV)*. Springer, 2022, pp. 193–218.

[17] 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)*. PMLR, 2018, pp. 5276–5285.

[18] 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*. Springer, 2020, pp. 537–542.

[19] W. Yang, X. Li, and Z. Zhang, "A regularized approach to sparse optimal policy in reinforcement learning," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[20] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," in *International Conference on Learning Representations*, 2021.

[21] L. Wu, J. Li, Y. Wang, Q. Meng, T. Qin, W. Chen, M. Zhang, T.-Y. Liu *et al.*, "R-drop: Regularized dropout for neural networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 10 890–10 905, 2021.

[22] H.-T. D. Liu, F. Williams, A. Jacobson, S. Fidler, and O. Litany, "Learning smooth neural functions via lipschitz regularization," in *ACM SIGGRAPH 2022 Conference Proceedings*, 2022, pp. 1–13.

[23] X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, and Q. Miao, "Deep reinforcement learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 4, pp. 5064–5078, 2022.

[24] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.