Training Verifiably Robust Agents Using Set-Based Reinforcement Learning

Anonymous authors
Paper under double-blind review

Revisions are marked in blue for convenience; the formatting will be reverted back to black in the final version.

Abstract

Reinforcement learning policies parametrized by deep neural networks have achieved strong performance for continuous control, yet even small input perturbations may lead to unpredictable behavior. This sensitivity limits their use in safety-critical domains, where formal verifiability and robustness guarantees are required. Our work addresses this gap between state-of-the-art adversarial training methods and formal verification to train verifiably robust agents. Previous works train networks with individual adversarial perturbations, making them only robust against the specific adversarial attacks used. In contrast, our approach propagates entire perturbed input sets, enclosing all possible adversarial attacks within a single network pass. We leverage this to explicitly penalize the size of the output set (minimizing closed-loop uncertainty) and thereby make the actor robust against all possible attacks. This is realized by the use of set-based policy gradients, where each output within the set has a different gradient, thereby balancing the accuracy and robustness of the network. Doing so, we achieve formal verifiability across different verification frameworks for up to 9 times larger input perturbations and improve certified worst-case performance, making them applicable in safety-critical environments.

1 Introduction

In recent years, deep reinforcement learning using neural networks has significantly improved solving complex control tasks (Mnih et al., 2015; OpenAI et al., 2020; Lillicrap et al., 2016). In many control tasks, state-action spaces are continuous, high-dimensional, and influenced by inherent system uncertainties, modeling errors, and sensor noise (Kober et al., 2013). In practice, we often do not have access to or knowledge about the underlying uncertainty distributions, but have an upper bound of the present perturbations. This poses an important challenge for reinforcement learning when parameterizing policies with neural networks, as they are sensitive to small input perturbations (Szegedy et al., 2014). This may lead to instabilities and safety violations of the controlled system: Fig. 1a shows an example of a navigation task where small input perturbations lead to trajectories that enter an unsafe set. Robustness to such perturbations is therefore essential. For deployment in safety-critical applications, it is additionally necessary to demonstrate safety and guaranteed worst-case performance. This necessitates formal verifiability under a bounded perturbation budget (Fig. 1b), which is not inherently guaranteed by current state-of-the-art robust reinforcement learning methods.

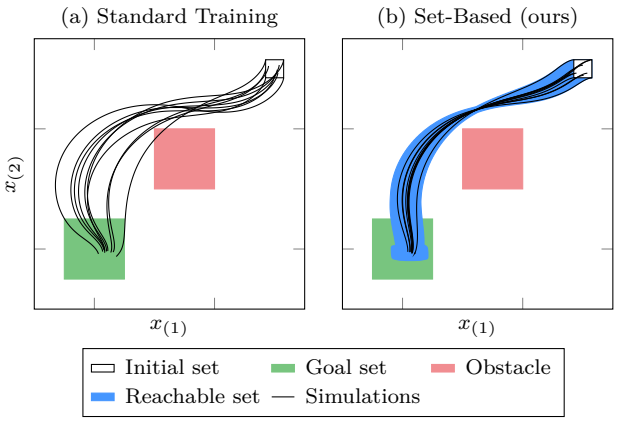

Figure 1: Comparison of standard and set-based reinforcement learning on a navigation task. (a) Trajectories of the standard agent intersect with the obstacle. (b) We can formally verify our robust agent.[1]

[1]Code: `https://t1p.de/ex9ov`, Video: `https://t1p.de/zjayi`

For the training of robust neural networks, a recent work proposes set-based training: For a set of possible inputs, the set of possible outputs is enclosed and the neural network is trained with a gradient set (Koller et al., 2025), i.e., each possible output has a different gradient. By picking gradients that point toward the center of the output set, the size of the output sets can be controlled. Thereby, the trained neural network is more robust and easier to formally verify as smaller propagated sets reduce the conservatism of the verification algorithm. In this work, we lift set-based training to reinforcement learning. Our main contributions are:

(i) A novel set-based reinforcement learning algorithm that, for the first time, computes a gradient set, which contains a different gradient for each possible output given input perturbations. Our algorithm trains agents that are provably more robust and formally verifiable across different available reachability-analysis frameworks.
(ii) A rigorous analysis of the underlying set propagation to derive a novel set-based loss function for regression tasks, which is used to compute gradient sets that optimize over entire output sets and thus achieve formal verifiability of the trained agents given input perturbations.
(iii) An extensive evaluation including a comparison with state-of-the-art adversarial training algorithms and ablation studies justifying our design choices.

2 Preliminaries

2.1 Notation

We write vectors as lowercase letters, matrices as uppercase letters, sets as calligraphic letters, and probability distributions as script font letters. The i-th entry of $v \in \mathbb{R}^n$ is written as $v_{(i)}$. The entry in the i-th row and j-th column of a matrix is $M_{(i,j)}$; $M_{(i,\cdot)}$ is the i-th row, and $M_{(\cdot,j)}$ the j-th column. The horizontal concatenation of matrices $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{n\times p}$ is denoted by $[A\ B]$. We write $|A|$ to denote the absolute values of each entry in A, and $\|v\|_p$ denotes the ℓ_p-norm. The identity matrix is denoted by $I_n \in \mathbb{R}^{n\times n}$, and the vector containing only ones or zeros is denoted by $\mathbf{1}$ or $\mathbf{0}$. The set of natural numbers up to $n \in \mathbb{N}$ is written as $[n] = \{1, 2, \dots, n\} \subset \mathbb{N}$. We denote a multidimensional interval by $\mathcal{I} = [l, u] = \{x \in \mathbb{R}^n \mid \forall i \in [n] \colon l_{(i)} \leq x_{(i)} \leq u_{(i)}\}$. Let $\mathcal{S} \subset \mathbb{R}^n$ be a set and $f \colon \mathbb{R}^n \to \mathbb{R}^m$ be a function, then $f(\mathcal{S}) = \{f(x) \mid x \in \mathcal{S}\}$. The gradient of a function f w.r.t. a variable x is denoted by $\nabla_x f(x, \cdot)$. The operator $\operatorname{diag} \colon \mathbb{R}^n \to \mathbb{R}^{n\times n}$ returns a diagonal matrix with the vector elements on its diagonal. The expected value of a random variable x under condition $y \sim \mathscr{Y}$ is $\mathbb{E}_{y\sim\mathscr{Y}}[x(y)]$.

2.2 Neural Networks

A feed-forward neural network $N_\theta \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}$ with learnable parameters θ consists of $\kappa \in \mathbb{N}$ alternating linear and activation layers, where the k-th layer has $n_k \in \mathbb{N}$ output neurons. The output $\hat{y} = N_\theta(x) \in \mathbb{R}^{n_\kappa}$ is computed by propagating an input $x \in \mathbb{R}^{n_0}$ through all layers.

Definition 1 (Neural Network, (Bishop & Nasrabadi, 2006, Sec. 5.1))**.** Given a neural network N_θ and an input $x \in \mathbb{R}^{n_0}$, the output $\hat{y} = N_\theta(x) \in \mathbb{R}^{n_\kappa}$ is given by

$$
\begin{aligned}
h_0 &= x, \\
h_k &= L_k(h_{k-1}) = \begin{cases} W_k\, h_{k-1} + b_k & \text{if } k\text{-th layer is linear,} \\ \sigma_k(h_{k-1}) & \text{otherwise,} \end{cases} \qquad \text{for } k \in [\kappa], \\
\hat{y} &= h_\kappa,
\end{aligned}
$$

with weights $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$, biases $b_k \in \mathbb{R}^{n_k}$, and elementwise activation functions $\sigma_k(\cdot)$.

2.3 Deep Deterministic Policy Gradient

We focus on continuous control tasks with a multidimensional state space $\mathcal{S}$ and action space $\mathcal{A}$ (Januszewski et al., 2021; Recht, 2019). Our set-based reinforcement learning approach is based on the deep deterministic policy gradient algorithm (DDPG) (Lillicrap et al., 2016) which consists of an actor $\mu_\phi \colon \mathcal{S} \to \mathcal{A}$ with

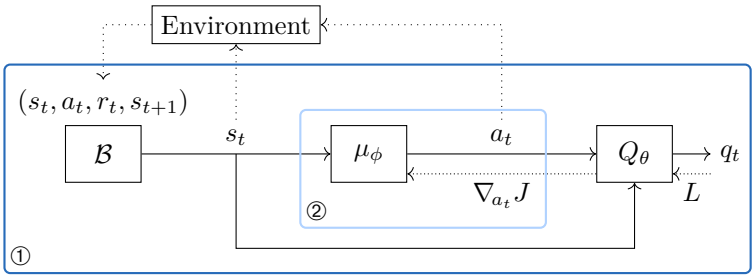

Figure 2: Illustration of the structure of the deep deterministic policy gradient algorithm; ① and ② show the components that are augmented by our set-based training (introduced in Sec. 3).

parameters ϕ and a critic $Q_\theta \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with parameters θ. Starting from an initial state s_0, the actor observes the state $s_t \in \mathcal{S}$ at time step t and returns an action $a_t = \mu_\phi(s_t)$, which controls the system until the next time step $t+1$. Using a reward function $r \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and the state transition probabilities $p(s_{t+1}|s_t, a_t)$, the environment returns a reward r_t and its next state s_{t+1} (Fig. 2). These transitions (s_t, a_t, r_t, s_{t+1}) are stored in a buffer $\mathcal{B}$ (Fig. 2). The training objective is to find the policy that maximizes the discounted cumulative reward (Silver et al., 2014, Eq. 8):

$$\max_\phi J(\phi, \rho^\mu) = \max_\phi \mathbb{E}_{s_t \sim \rho^\mu} \left[\sum_{t=0}^{\infty} \gamma^t \, r(s_t, \mu_\phi(s_t)) \right], \tag{1}$$

with discount factor $\gamma \in [0, 1]$ and discounted state visitation distribution ρ^μ for policy μ (Lillicrap et al., 2016, Sec. 2). The critic Q_θ approximates the expected total discounted reward for action a_t in state s_t (Lillicrap et al., 2016, Eq. 3):

$$Q_\theta(s_t, a_t) = r(s_t, a_t) + \gamma \, \mathbb{E}_{s_{t+1} \sim \rho^\mu} [Q_\theta(s_{t+1}, \mu_\phi(s_{t+1}))]. \tag{2}$$

The critic is trained off-policy with a stochastic policy β (Lillicrap et al., 2016, Eq. 4) and targets y_t (Lillicrap et al., 2016, Eq. 5) using the following bootstrapped Q-iterations:

$$L(\theta) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta} \big[(Q_\theta(s_t, a_t) - y_t)^2 \big], \tag{3}$$

$$y_t = r(s_t, a_t) + \gamma \, Q_\theta(s_{t+1}, \mu_\phi(s_{t+1})). \tag{4}$$

The actor is trained using the policy gradient (Lillicrap et al., 2016, Eq. 6):

$$\nabla_\phi J(\phi, \rho^\beta) \approx \mathbb{E}_{s_t \sim \rho^\beta} \Big[\nabla_{a_t} Q_\theta(s_t, a_t) \big|_{a_t = \mu_\phi(s_t)} \nabla_\phi \mu_\phi(s_t) \Big]. \tag{5}$$

2.4 Set-Based Computing

We model uncertainties using zonotopes, a continuous set representation, due to their favorable computational complexity of the required operations when propagating various sets through the neural networks.

Definition 2 (Zonotope (Girard, 2005))**.** Given center $c \in \mathbb{R}^n$ and generators $G \in \mathbb{R}^{n \times q}$, we define

$$\mathcal{Z} = \langle c, G \rangle_Z = \{ c + G \, \beta \mid \beta \in [-1, 1]^q \} \subset \mathbb{R}^n.$$

The affine map $x \mapsto A \, x + b$ of a zonotope $\mathcal{Z} = \langle c, G \rangle_Z$ is computed by (Althoff, 2010, Sec. 2.4)

$$A \, \mathcal{Z} + b = \{ A \, x + b \,|\, x \in \mathcal{Z} \} = \langle A \, c + b, A \, G \rangle_Z. \tag{6}$$

The enclosing interval of the zonotope $\mathcal{Z} = \langle c, G \rangle_Z$, i.e., $\mathcal{Z} \subseteq [l, u]$ and its diameter $\mathrm{dia}(\mathcal{Z})$, are calculated by (Althoff, 2010, Prop. 2.2)

$$l = c - |G| \mathbf{1}, \quad u = c + |G| \mathbf{1}, \quad \mathrm{dia}(\mathcal{Z}) := u - l = 2 |G| \mathbf{1}. \tag{7}$$

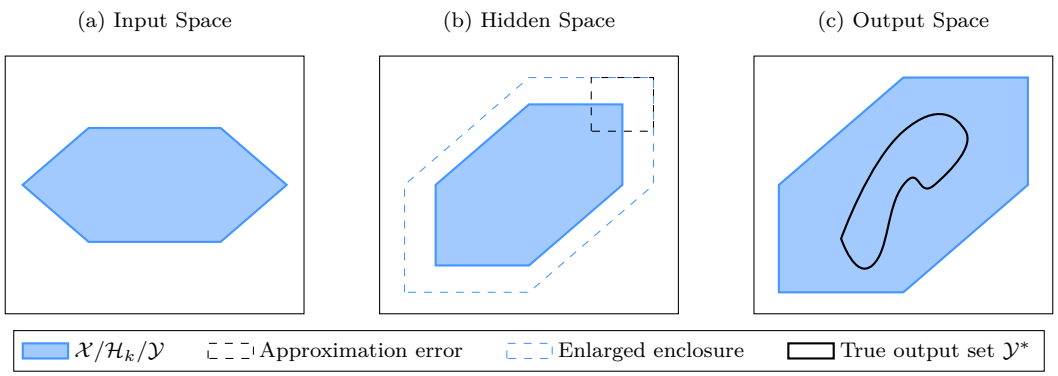

Figure 3: Set-based computing enables us to compute an output set enclosure. Full example in Appendix A.

For some derivations, we write $\ln\mathrm{Dia}(G) \coloneqq \ln(2\,|G|\mathbf{1})$ and $\ln\mathrm{Dia}'(G) \coloneqq \nabla_G \ln\mathrm{Dia}(G) = \mathrm{diag}(|G|\mathbf{1})^{-1}\,\mathrm{sign}\,G$ to avoid clutter, where $\mathrm{sign}(G)$ returns the sign of each entry of matrix G. The Minkowski sum of a zonotope $\mathcal{Z} = \langle c, G\rangle_Z$ and an interval $\mathcal{I} = [l, u]$ is computed by (Althoff, 2010, Prop. 2.1 and Sec. 2.4):

$$\begin{aligned}\mathcal{Z} \oplus \mathcal{I} &= \{x_1 + x_2 \mid x_1 \in \mathcal{Z},\, x_2 \in \mathcal{I}\} \\ &= \langle c + {}^1\!/\!_2(u + l), [G \;\; \mathrm{diag}({}^1\!/\!_2(u - l))]\rangle_Z.\end{aligned} \tag{8}$$

2.5 Set-Based Training of Neural Networks

The set-based training of a neural network (Koller et al., 2025) lifts the standard training to a set-based manner, enabling us to (i) propagate the modeled uncertainties $\mathcal{X} \subset \mathbb{R}^{n_0}$ through the neural network and (ii) define a loss function over the computed output set, and (iii) backpropagate the gradient set obtained from the loss function to update the weights of the neural networks. This trains neural networks to be more robust against such input uncertainties.

Computing the exact output set $\mathcal{Y}^* = N_\theta(\mathcal{X}) \subset \mathbb{R}^{n_\kappa}$ of a neural network for $\mathcal{X}$ is NP-hard (Katz et al., 2017). Thus, typically an enclosure of the output set $\mathcal{Y} \supseteq \mathcal{Y}^*$ is computed to remain computationally feasible. This is realized by conservatively propagating the input set through all layers of the neural network using the operations defined in Sec. 2.4, which we also visualize in Fig. 3.

Proposition 1 (Set-Based Forward Propagation (Singh et al., 2018))**.** Given an input set $\mathcal{X}$, the output set of a neural network can be enclosed as:

$$\begin{aligned}\mathcal{H}_0 &= \mathcal{X}, \\ \mathcal{H}_k &= \texttt{enclose}(L_k, \mathcal{H}_{k-1}), \quad \text{for } k \in [\kappa], \\ \mathcal{Y}^* \subseteq \mathcal{Y} &= \mathcal{H}_\kappa.\end{aligned}$$

We also write $\texttt{enclose}(N_\theta, \mathcal{X}) = \mathcal{Y}$ for the output enclosure of the entire network.

Given $\mathcal{Y}$ and a target $t \in \mathbb{R}^{n_\kappa}$, we can define a loss function $L(t, \mathcal{Y})$ over all points in $\mathcal{Y}$ (Koller et al., 2025, Sec. 3.1). Then, $\mathcal{G}_\mathcal{Y} = \nabla_\mathcal{Y} L(t, \mathcal{Y}) \subset \mathbb{R}^{n_\kappa}$ gives us a set of gradients for all points in $\mathcal{Y}$. This gradient set has the same shape as $\mathcal{Y}$, i.e., equal dimensionality and equal number of generators. Fig. 4 visualizes an example of such a gradient set, where a standard accuracy loss to move the output toward the target is combined with a robustness loss to penalize large – and therefore less robust – output sets. Analogous to Prop. 1, this gradient set can then be propagated backward through the network to obtain a gradient set $\mathcal{G}_k$ at each layer $k \in [\kappa]$.

Proposition 2 (Set-Based Backward Propagation (Koller et al., 2025, Sec. 4.2))**.** Given a gradient set $\mathcal{G}_\mathcal{Y} \subset \mathbb{R}^{n_\kappa}$ computed with respect to an output set $\mathcal{Y}$ of a neural network, the gradient sets with respect to

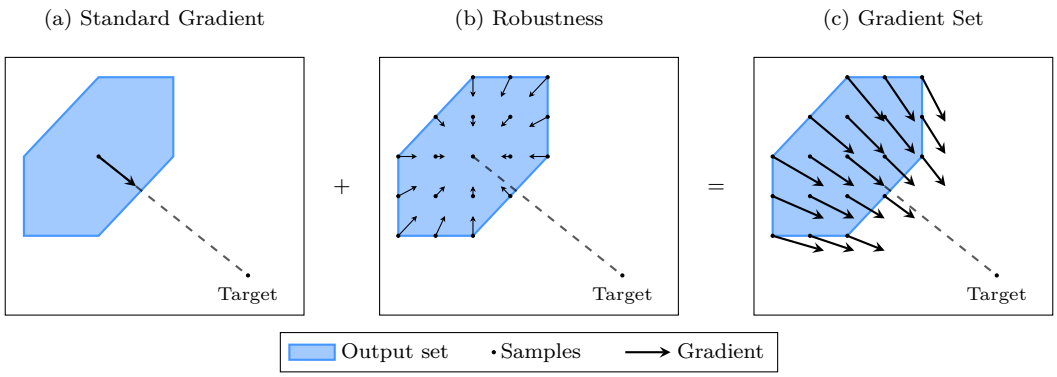

Figure 4: Visualization of a gradient set, where each point in the output set has a different gradient. Here, a penalty on the size of the output set moves each point in the set closer to the center, resulting together with the standard accuracy loss in a more robust network.

each layer of the neural network can be enclosed as:

$$\begin{aligned} \mathcal{G}_\kappa &= \mathcal{G}_\mathcal{Y}, \\ \mathcal{G}_{k-1} &= \texttt{backpropEnclose}(L_k, \mathcal{G}_k), \quad \text{for } k \in [\kappa], \\ \mathcal{G}_\mathcal{X} &= \mathcal{G}_0. \end{aligned}$$

We also write $\texttt{backpropEnclose}(N_\theta, \mathcal{G}_\mathcal{Y}) = \mathcal{G}_\mathcal{X}$ for the gradient set with respect to the input set $\mathcal{X}$.

These gradient sets are typically computed using information obtained from the forward pass, and can subsequently be used to update the parameters of each layer of the neural network (Koller et al., 2025, Prop. 14). As all operations are computed using basic matrix manipulations, these can be efficiently implemented on a GPU in a batch-wise manner.

2.6 Problem Statement

We are given a reinforcement learning task with uncertain initial states $s_0 \in \mathcal{S}_0 \subset \mathbb{R}^n$. Moreover, we model state uncertainties with an ℓ_∞-ball of radius $\epsilon \in \mathbb{R}_+$. The set of all perturbations is $\mathcal{V}_\epsilon^\infty := \{\nu \colon \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n \mid \forall s_t \in \mathbb{R}^n, \forall t \in \mathbb{R}_+ \colon \|\nu(s_t, t) - s_t\|_\infty \leq \epsilon\}$. Our goal is to robustly maximize the reward under adversarial perturbations

$$\max_\phi \min_{\nu \in \mathcal{V}_\epsilon^\infty} J(\phi, \rho^{\mu \circ \nu}), \tag{9}$$

where $\rho^{\mu \circ \nu}$ denotes the state visitation distribution perturbed with adversary $\nu \in \mathcal{V}_\epsilon^\infty$. Further, let $\mathcal{R}$ enclose all reachable states for the time interval $[0, t_{\text{end}}]$ (Althoff, 2010). We derive a verified performance as a lower bound on the reward by evaluating the worst trajectory from $\mathcal{R}$.

3 Set-Based Reinforcement Learning

We present a novel approach for training verifiably robust actors using set-based policy gradients. This approach lifts set-based training to reinforcement learning, enabling us to explicitly control the geometry in both size and shape of the policy output set by penalizing undesirable directions and magnitudes. As a result, the conservativeness induced when propagating these sets through the network (Prop. 1) is reduced. This stands in stark contrast to existing robust reinforcement learning approaches, which rely on pointwise perturbations or adversarial samples and thereby never expose the network to the full range of perturbations.

Set-based actor and critic. We apply this set-based policy gradient computation to both the actor and the critic (*SA-SC*), which corresponds to a set-based evaluation of ① in Fig. 2. The main steps are given

Algorithm 1 Set-based reinforcement learning.

Randomly initialize Q_θ, μ_ϕ with θ, ϕ
Initialize replay buffer $\mathcal{B}$
for episode $= 1, \ldots,$ maxEpisodes **do**
  Get initial observation s_0
  for $t = 0, 1, \ldots,$ maxSteps **do**
    `(i) Fill replay buffer` $\mathcal{B}$ `--`
    $\mathcal{S}_t \leftarrow \langle s_t, \epsilon I \rangle_Z$ ▷ Eq. 10
    $\mathcal{A}_t \leftarrow \texttt{enclose}(\mu_\phi, \mathcal{S}_t)$ ▷ Eq. 11
    $\tilde{\mathcal{A}}_t \leftarrow \mathcal{A}_t + e_t,\ e_t \sim \mathbb{P}(E)$ ▷ Eq. 12
    $r_t \leftarrow r(s_t, c_{\tilde{\mathcal{A}}_t})$
    $s_{t+1} \leftarrow$ execute action $c_{\tilde{\mathcal{A}}_t}$
    Store transition $(s_t, \tilde{\mathcal{A}}_t, r_t, s_{t+1})$ in $\mathcal{B}$
    `(ii) Train networks --`
    Sample n transitions from $\mathcal{B}$
    Compute targets y_i using Q_θ ▷ Eq. 3
    Update critic Q_θ ▷ Prop. 3
    Update actor μ_ϕ ▷ Def. 4

in Alg. 1. As in standard reinforcement learning, Alg. 1 starts by (i) filling up the replay buffer $\mathcal{B}$: Given the current state s_t, we enclose all possible adversaries with a zonotope

$$\mathcal{S}_t = \{\nu(s_t, t) \mid \nu \in \mathcal{V}_\epsilon^\infty\} = \langle s_t, \epsilon\, I \rangle_Z, \tag{10}$$

and enclose the set of possible actions by propagating $\mathcal{S}_t$ through the actor (Prop. 1):

$$\mathcal{A}_t = \langle c_{\mathcal{A}_t}, G_{\mathcal{A}_t} \rangle_Z = \texttt{enclose}(\mu_\phi, \mathcal{S}_t). \tag{11}$$

For the off-policy training of the critic, we store the state and action zonotopes $\mathcal{S}_t$ and $\mathcal{A}_t$ in the replay buffer $\mathcal{B}$ and compute the corresponding set of critic outputs:

$$\mathcal{Q}_t = \langle c_{\mathcal{Q}_t}, G_{\mathcal{Q}_t} \rangle_Z = \texttt{enclose}(Q_\theta, \mathcal{S}_t \times \tilde{\mathcal{A}}_t), \tag{12}$$

where the Cartesian product ($\times$) respects the dependencies between two zonotopes (Lützow & Althoff, 2023, Sec. II.C). The environment receives the center of the action set $c_{\tilde{\mathcal{A}}_t} := c_{\mathcal{A}_t}$ and returns the reward $r(s_t, c_{\tilde{\mathcal{A}}_t})$ and the next state s_{t+1}, which are stored in the replay buffer $\mathcal{B}$ as transition $(s_t, \tilde{\mathcal{A}}_t, r_t, s_{t+1})$.

Subsequently, part (ii) starts by training the critic. We randomly sample n transitions from the buffer $\mathcal{B}$. For each transition $i \in [n]$, we compute the target y_i using Eq. 3 and compute a continuous set of gradients for the entire input set using a set-based loss function:

Proposition 3 (Set-Based Regression Loss)**.** Given output set $\mathcal{Q}_i = \langle c_{\mathcal{Q}_i}, G_{\mathcal{Q}_i} \rangle_Z \subset \mathbb{R}$ and target $y_i \in \mathbb{R}$, the set-based regression loss is defined as

$$L_{Reg}(y_i, \mathcal{Q}_i) = \underbrace{{}^1\!/_2 (c_{\mathcal{Q}_i} - y_i)^2}_{\text{standard training loss}} + \underbrace{{}^{\eta_Q}\!/_\epsilon \ \mathrm{lnDia}(G_{\mathcal{Q}_i})}_{\text{robustness}},$$

with weighting factor $\eta_Q \in \mathbb{R}_+$ and perturbation radius $\epsilon \in \mathbb{R}_+$. The gradient of L_{Reg} w.r.t. $\mathcal{Q}_i$ is:

$$\nabla_{\mathcal{Q}_i} L_{Reg}(y_i, \mathcal{Q}_i) = \left\langle c - y_i, {}^{\eta_Q}\!/_\epsilon \ \mathrm{lnDia}'(G_{\mathcal{Q}_i}) \right\rangle_Z.$$

Proof. The gradient w.r.t. a zonotope is represented as a zonotope as well, consisting of the gradient w.r.t. the center and the gradient w.r.t. the generator matrix (Koller et al., 2025, Def. 8). Hence, the gradient of

the set-based regression is computed by:

$$\begin{aligned}\nabla_{\mathcal{Q}_i} L_{Reg}(y_i, \mathcal{Q}_i) &= \left\langle \nabla_{c_{\mathcal{Q}_i}} L_{Reg}(y_i, \mathcal{Q}_i), \nabla_{G_{\mathcal{Q}_i}} L_{Reg}(y_i, \mathcal{Q}_i) \right\rangle_Z \\ &= \left\langle c_{\mathcal{Q}_i} - y_i, \frac{\eta_Q}{\epsilon} \operatorname{diag}(|G_{\mathcal{Q}_i}|\,\mathbf{1})^{-1} \operatorname{sign} G_{\mathcal{Q}_i} \right\rangle_Z \\ &= \left\langle c_{\mathcal{Q}_i} - y_i, \frac{\eta_Q}{\epsilon} \operatorname{lnDia}'(G_{\mathcal{Q}_i}) \right\rangle_Z. \end{aligned}$$

□

The set-based regression loss combines an accuracy term with a robustness loss term: The accuracy term is computed by the half-squared error (Bishop & Nasrabadi, 2006, Eq. 5.14) between the center and the target, thereby improving the accuracy of the network as in regular (point-wise) reinforcement learning. The robustness term improves the verifiability by shrinking the computed enclosure. This is realized by penalizing the volume of the enclosed set (Eq. 7). Together, these terms result in a set of gradients, such that each point in the set gets pulled towards the target, as visualized by the gradient set in Fig. 4.

Finally, to train the actor, this combined loss is extended using a set-based policy gradient. As for the critic update, the loss consists of two terms: the accuracy term, which points to policy improvement, and the robustness term, which penalizes the volume of the enclosed set. To update the weights of the actor, we apply Prop. 2 twice, first on the critic network to backpropagate the gradient set to the action space, and subsequently through the actor network to update all weights. Intuitively, this results in states in the local neighborhood captured by the perturbations behaving similarly. Thus, their action set can be captured using a small enclosure, thereby improving verifiability as conservativeness induced by larger sets is reduced.

Definition 3 (Set-Based Policy Gradient (*SA-SC*))**.** Given states $\mathcal{S}_i$ with the corresponding actions $\mathcal{A}_i = \langle c_{\mathcal{A}_i}, G_{\mathcal{A}_i} \rangle_Z$ and critic outputs $\mathcal{Q}_i = \langle c_{\mathcal{Q}_i}, G_{\mathcal{Q}_i} \rangle_Z$, a set-based policy gradient is defined as

$$\begin{aligned} \nabla_{\mathcal{A}_i} J_{Set}(\mu_\phi) &\coloneqq \langle \underbrace{\nabla_{c_{\mathcal{A}_i}} J_{Set}(\mu_\phi)}_{\text{policy gradient}}, \underbrace{\nabla_{G_{\mathcal{A}_i}} J_{Set}(\mu_\phi)}_{\text{robustness}} \rangle_Z, \\ \text{where } \nabla_{c_{\mathcal{A}_i}} J_{Set}(\mu_\phi) &= \mathbb{E}_{s_i \sim \rho^\beta} \left[\nabla_{c_{\mathcal{A}_i}} c_{\mathcal{Q}_i} \right], \\ \text{and } \nabla_{G_{\mathcal{A}_i}} J_{Set}(\mu_\phi) &= -\eta_\mu / \epsilon \, \mathbb{E}_{s_i \sim \rho^\beta} \left[\omega \operatorname{lnDia}'(G_{\mathcal{A}_i}) + (1-\omega) \nabla_{G_{\mathcal{A}_i}} \operatorname{lnDia}'(G_{\mathcal{Q}_i}) \right], \end{aligned}$$

with weights $\eta_\mu \in \mathbb{R}_+$, $\omega \in [0, 1]$ and perturbation $\epsilon \in \mathbb{R}_+$.

In more detail, the set-based policy gradient itself is a zonotope consisting of a center, which corresponds to the standard policy gradient in Eq. 5, and a generator matrix. The generator matrix $\nabla_{G_{\mathcal{A}_i}} J_{Set}(\mu_\phi)$ uses the factor ω to weight the robustness loss for the action set $\mathcal{A}_i$ and the gradients of the robustness loss of the critic outputs $\mathcal{Q}_i$ in the action space (Fig. 2). Finally, given the gradients w.r.t. the output of the actor and the critic, we can update the respective parameters as described in Sec. 2.5.

Proposition 4 (Runtime Complexity (*SA-SC*))**.** Given an actor μ and a critic Q with $n_{\max,\mu}$, $n_{\max,Q}$ being the maximum number of neurons per layer, and $n_{\text{total},\mu}$, $n_{\text{total},Q}$ the total number of neurons of the actor and critic, respectively, then the runtime complexity per sample is polynomial and given by

$$\mathcal{O}\big(n_{\max,\mu}^2 n_{\text{total},\mu} \kappa_\mu + n_{\max,Q}^2 (n_{\text{total},\mu} + n_{\text{total},Q}) \kappa_Q\big).$$

As all operations are based on basic matrix manipulations, it can be efficiently implemented on a GPU in a batch-wise manner.

Proof. From Sec. 2.5 and the runtime complexity analysis in (Koller et al., 2025, Prop. 17), we observe that the complexity is dominated by the forward and backward propagation of the zonotope, and in particular by the affine maps (Def. 1, Eq. 6). In each nonlinear layer k, n_k generators get added to the current zonotope through Eq. 7. Thus, we upper-bound the propagation through the actor μ with $\mathcal{O}(n_{\max,\mu}^2 n_{\text{total},\mu} \kappa_\mu)$. This upper bound can also be used for the propagation through the critic, with the exception that the input zonotope to the critic already has all generators accumulated by the actor propagation, and thus gives us $\mathcal{O}(n_{\max,Q}^2 (n_{\text{total},\mu} + n_{\text{total},Q}) \kappa_Q)$. □

Optimizing runtime. Please note that for $\omega = 1$, the set-based gradient simplifies, and we can omit the set propagation through the critic, since the gradients are solely computed based on the robustness loss of the action set (*SA-PC*). In this case, we only require a set-based evaluation of the actor (② of Fig. 2), and the runtime complexity is thus reduced to $\mathcal{O}(n^2_{\max,\mu} n_{\text{total},\mu} \kappa_\mu)$.

Definition 4 (Set-Based Policy Gradient (*SA-PC*))**.** Given states $\mathcal{S}_i$ with the corresponding actions $\mathcal{A}_i = \langle c_{\mathcal{A}_i}, G_{\mathcal{A}_i} \rangle_Z$, a set-based policy gradient is defined as

$$\nabla_{\mathcal{A}_i} J_{Set}(\mu_\phi) \coloneqq \langle \underbrace{\nabla_{c_{\mathcal{A}_i}} J_{Set}(\mu_\phi)}_{\text{policy gradient}}, \underbrace{\nabla_{G_{\mathcal{A}_i}} J_{Set}(\mu_\phi)}_{\text{robustness}} \rangle_Z,$$

$$\text{where } \nabla_{c_{\mathcal{A}_i}} J_{Set}(\mu_\phi) = \mathbb{E}_{s_i \sim \rho^\beta} \left[\nabla_{c_{\mathcal{A}_i}} Q_\theta(s_i, c_{\mathcal{A}_i}) \right],$$

$$\nabla_{G_{\mathcal{A}_i}} J_{Set}(\mu_\phi) = -\eta_\mu / \epsilon \, \mathbb{E}_{s_i \sim \rho^\beta} \left[\ln\text{Dia}'(G_{\mathcal{A}_i}) \right],$$

with weight $\eta_\mu \in \mathbb{R}_+$ and perturbation $\epsilon \in \mathbb{R}_+$.

We believe that this novel view of reinforcement learning using set-based computing is a major step toward verifiability, as subsequent verification steps are directly brought into the training process. There is also a connection to the probabilistic view, prevalent in reinforcement learning theory, which we discuss and relate to our set-based view in Appendix B.

4 Evaluation

We use the MATLAB toolbox CORA (Althoff, 2015; Althoff et al., 2025) to implement our novel set-based reinforcement learning algorithm and compare the *SA-PC* and the *SA-SC* implementation against standard (point-based) training (*PA-PC*) and three state-of-the-art adversarial methods: *Naive*-, *Grad*- and *MAD* (Maximum Action Difference)-based implementations (Pattanaik et al., 2018, Alg. 2 and 4)(Zhang et al., 2020), which compute adversarial attacks to approximate the worst-case observation within a perturbation set. Additionally, we also compare our set-based approach with *RORL* (Yang et al., 2022), an adversarial smoothing based approach, which has been primarily developed for offline RL, but can also be used in online RL to mitigate out of distribution perturbations. We do not compare *RORL* with our TD3 implementation, since *RORL* is already a critic ensemble based method itself. We consider four diverse benchmarks in our evaluation taken from previous work on robust reinforcement learning (Yuan et al., 2022; Krasowski et al., 2023): *Navigation Task*, *1D Quadrotor*, *Inverted Pendulum*, and *2D Quadrotor*. A detailed description of all benchmarks can be found in Appendix C. Also following these works, we use neural networks with ReLU activations and two hidden layers of 64 and 32 neurons for the actor and critic networks. The output layer of the actor has a tanh activation. We provide the mean results and the 95% confidence interval across five different random seeds. Hyperparameters and evaluation details are given in Appendix C.

Verified performance. Following our problem statement (Sec. 2.6), we evaluate our agents based on their worst-case cumulative reward given uncertainties, which we compute using reachability analysis in CORA. Reachability analysis bounds the evolution of the uncertainty through the entire dynamic system until the time horizon t_{end} is reached. More formally, starting at an initial point s_0, reachability analysis encloses all reachable states within a time interval $[0, t_{\text{end}}]$ with time horizon $t_{\text{end}} \in \mathbb{R}_+$ (Fig. 1b). Additional uncertainties (Eq. 10) are added at each time step $t \in \{0, 1, \dots, t_{\text{end}}\}$ and carried through until the time horizon is reached. Please note that, in general, this process is outer-approximative due to the continuous time and nonlinearities within the system and networks. Thus, reachability analysis obtains a formally verified lower bound of the worst-case cumulative reward, which we refer to as *verified performance* from now on. In particular, for reward functions of the form $r(s_t, a_t) = w^\top |s_t - s^*|$, we can use set-based computing to obtain the verified performance $\underline{V}_\mu(s_0)$ computed from the set of states $\mathcal{S}_t = \langle c_t, G_t \rangle_Z$ obtained by CORA:

$$\underline{V}_\mu(s_0) = \sum_{t=0}^{t_{\text{end}}} \gamma^t \max_{s_t \in \mathcal{S}_t} w^\top |s_t - s^*| \overset{\text{Eq. 6, Eq. 7}}{=} \sum_{t=0}^{t_{\text{end}}} \gamma^t \big(w^\top |c_t - s^*| + \text{dia}(w^\top \mathcal{S}_t)/2 \big). \tag{13}$$

Intuitively, $\underline{V}_\mu(s_0)$ is high if we can formally verify the robustness of our agent and drops otherwise.

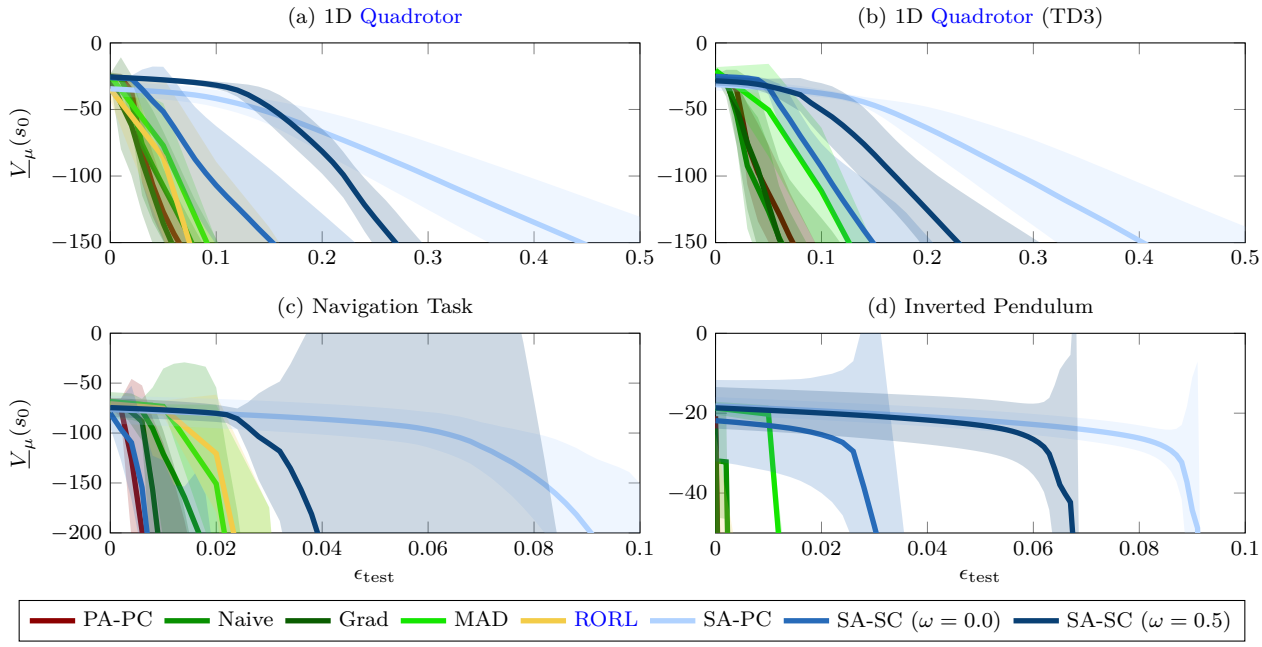

Figure 5: Comparison of verified performance $\underline{V}_\mu(s_0)$ (↑) for the (a) *1D Quadrotor*, (c) *Navigation Task*, and (d) *Inverted Pendulum*. The TD3 implementation is compared in (b) for the *1D Quadrotor*.

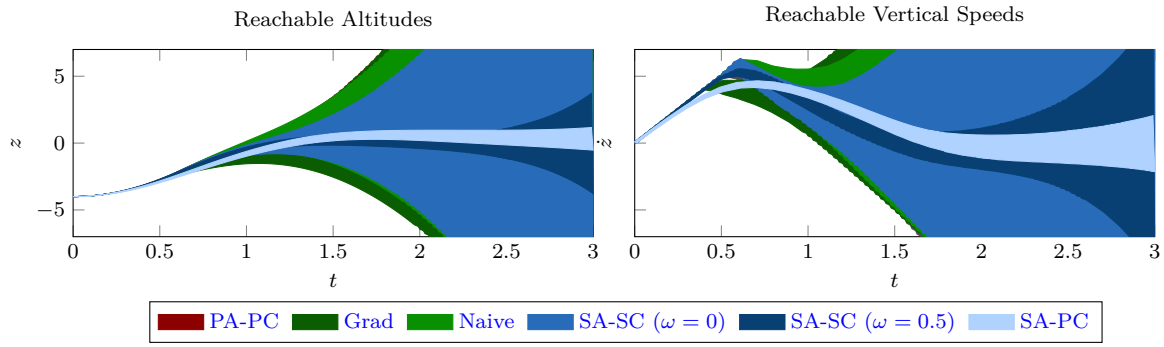

Figure 6: *Quadrotor 1D*: Comparison of reachable altitudes and vertical speeds for $\epsilon_{\text{test}} = 0.15$.

Main results. We present the verified performance with increasing perturbation radius ϵ_{test} (Eq. 10) of the considered training methods in Fig. 5. The set-based algorithms *SA-PC* and *SA-SC* train agents that can be verified for up to 9 times larger perturbation radii ϵ_{test} than the other methods. As ϵ_{test} increases, the set-based training methods show a higher verified performance across all benchmarks, indicating reduced performance degradation under growing disturbance (Fig. 5). Notably, the agents are even robust for perturbations above the trained perturbation radius $\epsilon_{\text{train}} = 0.1$ (Tab. 3) on the quadrotor. We showcase the improved robustness in Fig. 6 using a large perturbation radius, where the reachable set of the *SA-PC* agent remains much smaller and thus the stability can be formally verified.

Ablation study. Let us continue with our ablation study on various components of our algorithm:

1) Influence of weighting factor ω (Def. 3): Recall that this parameter is used to determine where the volume of the set is penalized: With $\omega = 0$, only the output set of the critic is penalized (*SA-SC*) and with $\omega = 1$, only the output set of the actor is penalized, which corresponds to the *SA-PC* method. The *SA-PC* implementation trains more conservative actors, which perform best for large ϵ_{test} while having a worse verified performance for small ϵ_{test}. For example, the *SA-PC* actor for the *1D Quadrotor* is the most robust but reaches the goal later using lower vertical velocities (Fig. 6). By fine-tuning ω, a balance can be

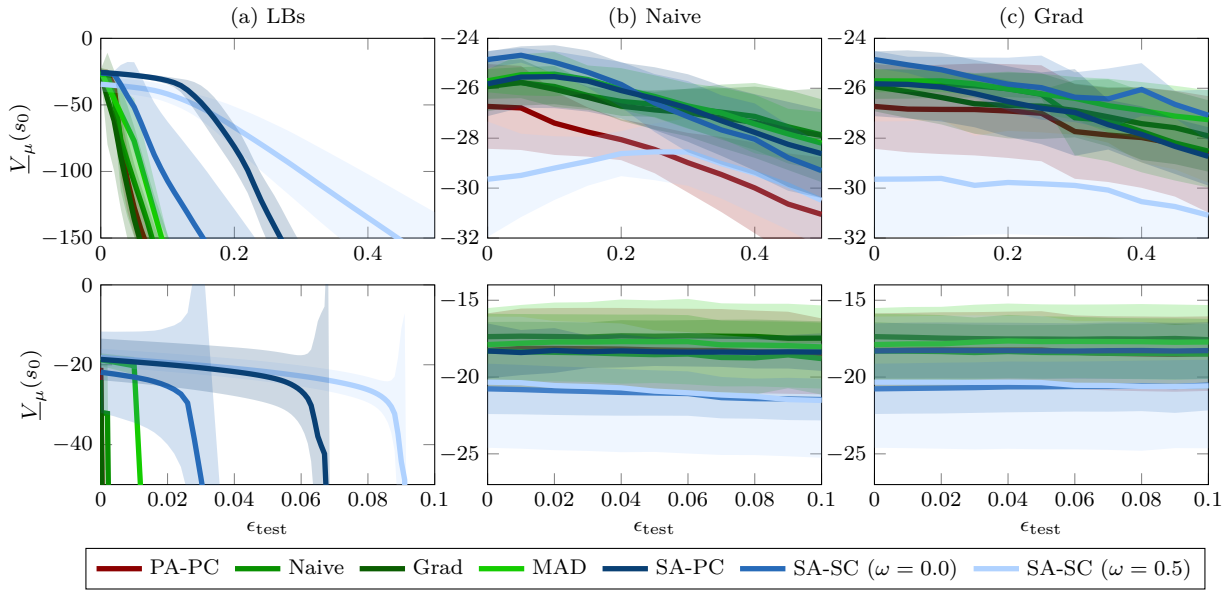

Figure 7: (Top) *Quadrotor 1D* and (bottom) *Inverted Pendulum*: Comparison of the return under *Naive* and *Grad* attacks with $\underline{V}_\mu(s_0)$ (↑).

Table 1: Time [s] for verification of Navigation task agents. Entries with − indicate that the verification toolbox is not able to verify the system.

Toolbox	CORA	CROWN-Reach	JuliaReach	NNV
Standard	423.45	130.12	–	–
Set-Based	1.99	20.65	4.49	1903.24

found where the verified performance for small ϵ_{test} can be regained while still being much more robust for larger ϵ_{test} (Fig. 5).

2) Performance under different attacks: It is well-known that networks trained using adversarial attacks are robust against their respective attack method but might not be robust against other methods. This ablation study confirms this observation: Under *Naive* and *Grad* attacks (Fig. 7b-c), also their respectively trained agents outperform standard point-based training with DDPG (Lillicrap et al., 2016). However, verified performance (Fig. 7a) captures all attacks and, thus, *Grad*, *Naive*, and *MAD* methods do not improve on standard point-based training (*PA-PC*) across all benchmarks (Fig. 5), while *SA-SC* and *SA-PC* demonstrate their *formal* robustness against the entire set of all possible perturbations.

Yet, previous works evaluated their agents solely through adversarial attacks, with learned probabilistic dynamic models (Yang et al., 2024) or randomized smoothing (Wu et al., 2022), which provide an empirical upper bound or a probabilistic bound of the worst-case reward. As demonstrated here, these are not sufficient when formal safety is demanded.

3) Performance under different verification methods: Since different formal verification techniques may already be in place for various safety-critical systems, we also show in Tab. 1 that our set-based trained agents can be verified using alternative methods that do not rely on zonotopes. All toolboxes (Althoff, 2015; Xiangru Zhong, 2024; Bogomolov et al., 2019; Tran et al., 2020) verified the set-based agent, whereas only two verified the standard-trained agent, taking much longer to do so. Hence, our set-based agents are easier to verify across different toolboxes due to their robust policy.

4) Runtime analysis: Set-based reinforcement learning can be efficiently computed batch-wise on a GPU, but the memory load remains challenging. The different runtimes for 10 learning epochs are listed in Tab. 2

Table 2: Training time [s/10 epochs] of *SA-PC* and *SA-SC* only show a moderate increase over baselines.

Benchmark	PA-PC	Naive	Grad	SA-PC	SA-SC
1D Quad.	1.58	2.10	1.98	2.77	7.92
Pendulum	1.82	2.04	2.02	2.87	7.66
Nav. Task	2.35	2.89	2.84	4.35	12.43

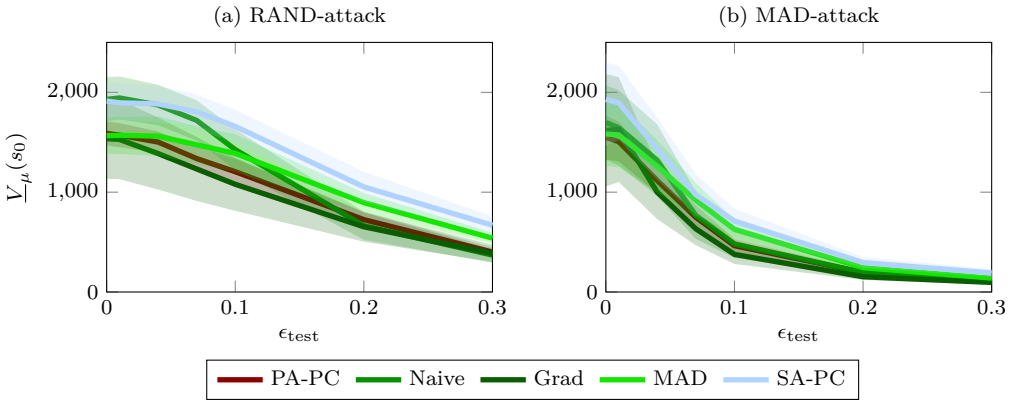

Figure 8: Comparison of $\underline{V}_\mu(s_0)$ for *Hopper-v2* approximated by (a) uniform random noise and (b) MAD (Zhang et al., 2020).

and were run on a server with two *AMD EPYC 7763* 64 core processors, 2 TB RAM, and an *NVIDIA A100-PCIE* 40 GB GPU. For *SA-SC*, the storing of entire action sets with many generators in the replay buffer $\mathcal{B}$ is memory-consuming and computationally more expensive. However, *SA-PC* emerges with both competitive training time and high verified performance (Fig. 5).

5) Extension to ensemble methods: Our proposed set-based reinforcement learning can be directly extended for the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) (Fujimoto et al., 2018) and other ensemble algorithms (Januszewski et al., 2021). Fig. 5b shows overall a similar performance as with DDPG and for the *SA-PC* version, a better performance for small perturbation radii ϵ_{test} of the set-based TD3 algorithm for the *1D Quadrotor*.

6) Scalability: In Fig. 8, we show that our algorithm scales to more complex benchmarks with intricate dynamics and larger state and action spaces (Todorov et al., 2012). While the worst-case reward cannot be directly evaluated for every benchmark, we show that for systems lacking formal verification toolboxes, our algorithm scales effectively in practice and empirically obtains robust neural networks. We discuss the implementation of the empirical attacks and the benchmark details in Appendix C.3.

7) Robustness tradeoff: Our set-based reinforcement learning implementations *SA-PC* and *SA-SC* trade off robustness and performance via the weighting factors η_μ and ω. Increasing η_μ increases robustness, while large ω values penalize the size of the action set. Additional experiments, hyperparameter ablations, and the entire learning history are provided in Appendix C.

5 Related Work

Related works on robust reinforcement learning (Zhang et al., 2021a; Huang et al., 2017; Deshpande et al., 2021; Mandlekar et al., 2017; Lütjens et al., 2020; Zhang et al., 2021b) propose a competitive framework with an adversary (Moos et al., 2022; Pinto et al., 2017): In observation-robust algorithms, the adversary exploits the sensitivity of neural networks to choose a worst-case observation for the policy (Moos et al., 2022). Computing the worst-case observation is often intractable (Madry et al., 2018). Consequently, a variety of naive, gradient-based (Pattanaik et al., 2018; Mandlekar et al., 2017; Huang et al., 2017),

learning-based (Zhang et al., 2021a), and convex relaxation methods (Zhang et al., 2020), have been proposed to approximate adversarial observations. For instance, gradient-based methods typically employ the Fast Gradient Sign Method (FGSM) to approximate the most adversarial input (Goodfellow et al., 2015). These adversarial attacks have also been used in conservative smoothing techniques, mainly for offline reinforcement learning, but can also be applied to mitigate out of distribution perturbations in an online learning setting Yang et al. (2022).On the contrary, our method does not rely on a single adversarial instance. Instead, it updates the agent parameters using the entire set of admissible perturbations, thereby improving robustness and enabling stronger formal guarantees of verifiability.

The safe deployment in safety-critical environments requires more than empirical robustness. It demands formal guarantees. While prior work often relies on adversarial attacks or randomized smoothing (Wu et al., 2022) to determine a probabilistic upper bound on the worst-case performance, these approaches do not provide a provable lower bound. In contrast, our work focuses on formal verification of neural network-based control policies. Recent advances (Manzanas Lopez et al., 2023; Brix et al., 2023) make it possible to verify entire neural network control systems, and if the obtained reachable set does not violate specifications, the neural network control system is verified as shown in Fig. 1(b).

6 Limitations

Training and verification. Our approach does not guarantee that the obtained agent is robust in all scenarios, and verification is still required at deployment. Moreover, the formal closed-loop verification of neural network controllers is itself an active research area, and existing toolboxes do not yet support the complex dynamics commonly used in deep reinforcement learning (e.g., MuJoCo benchmarks). For such systems, we can demonstrate empirical robustness improvements through adversarial attacks (Fig. 8), but computing the provable lower bound on the worst-case return is currently out of reach.

Training runtime and memory. While the per-sample training cost of our algorithm is polynomial (Prop. 4), it is more computationally expensive compared to standard training due to the repeated zonotope propagations. This effect is most pronounced for *SA-SC*, where the sets need to be propagated through both the actor and the critic. The runtime measurements in Tab. 2 reflect this: *SA-SC* is roughly 3–5 times slower than the point-based baselines, whereas the *SA-PC* variant only incurs a moderate overhead. We believe such runtime increases are acceptable given the substantially more robust networks we obtain.

Robustness-performance trade-off. Our method suffers from the well-known trade-off between accuracy and robustness. For example, *SA-PC* ($\omega = 1$) yields the highest verified performance under large perturbations ϵ_{test}, but produces more conservative policies that can underperform the point-based baseline for small or zero perturbations – e.g., the *1D Quadrotor* agent reaches the goal more slowly.

Modeling the system and uncertainties. The verification assumes white-box access to the system dynamics and bounded uncertainties. While we obtain safety guarantees for the assumed model, we cannot show safety if there is a mismatch between the model and the real system.

7 Conclusion

We introduce the first set-based reinforcement learning algorithm. Unlike other algorithms that rely on adversarial inputs, our approach trains the networks with entire sets of inputs and gradients. As demonstrated by our experiments, this unique advantage makes the resulting networks *verifiably* robust, and does not just improve empirical performance. In contrast, we show that algorithms relying on adversarial inputs only improve against their adversarial attack method. Thus, our trained controllers can be formally verified for large perturbation sets, which is essential for their deployment in safety-critical environments. Consequently, set-based reinforcement learning is an effective, novel approach for training robust neural network controllers. We also show that our set-based learning approach is not limited to the exact setup considered in our work and also generalizes to, e.g., ensemble methods such as TD3. Thus, we believe that our work sparks many research directions to train robust agents that can be deployed to safety-critical systems.

References

Matthias Althoff. *Reachability analysis and its application to the safety assessment of autonomous cars*. PhD thesis, Technical University of Munich, 2010.

Matthias Althoff. An introduction to CORA 2015. In *Proc. of Int. Workshop on Applied Verification of Continuous and Hybrid Systems*, pp. 120–151, 2015.

Matthias Althoff, Niklas Kochdumper, Tobias Ladner, Maximilian Perschl, and Mark Wetzlinger. CORA manual. *Technical University of Munich*, 2025. URL `https://cora.in.tum.de/manual`.

Chi Au and Judy Tam. Transforming variables using the dirac generalized function. *The American Statistician*, 53(3):270–272, 1999.

Christopher M. Bishop and Nasser M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. JuliaReach: A toolbox for set-based reachability. In *Proc. of the Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, pp. 39–44. Association for Computing Machinery, 2019.

Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The fourth int. verification of neural networks competition (VNN-COMP 2023): Summary and results, 2023.

Aditya M. Deshpande, Ali A. Minai, and Manish Kumar. Robust deep reinforcement learning for quadcopter control. *Int. Federation of Automatic Control (IFAC-PapersOnLine)*, 54(20):90–95, 2021.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2018.

Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Proc. of the Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, pp. 291–305, 2005.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2015.

Geoffrey E Hinton and Zoubin Ghahramani. Generative models for discovering sparse distributed representations. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 352(1358): 1177–1190, 1997.

Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *Proc. Int. Conf. on Learning Representations (ICLR), Workshop Track*, 2017.

Piotr Januszewski, Mateusz Olko, Michał Królikowski, Jakub Swiatkowski, Marcin Andrychowicz, Łukasz Kuciński, and Piotr Miłoś. Continuous control with ensemble deep deterministic policy gradients. In *Deep RL Workshop NeurIPS*, 2021.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proc. of the Int. Conf. on Computer Aided Verification (CAV)*, pp. 97–117, 2017.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The Int. Journal of Robotics Research*, 32(11):1238–1274, 2013.

Lukas Koller, Tobias Ladner, and Matthias Althoff. Set-based training for neural network verification. *Transactions on Machine Learning Research*, 2025.

Hanna Krasowski, Jakob Thumm, Marlon Müller, Lukas Schäfer, Xiao Wang, and Matthias Althoff. Provably safe reinforcement learning: Conceptual analysis, survey, and benchmarking. *Transactions on Machine Learning Research*, 2023.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2016.

Diego Manzanas Lopez, Matthias Althoff, Luis Benet, Xin Chen, Jiameng Fan, Marcelo Forets, Chao Huang, Taylor T. Johnson, Tobias Ladner, Wenchao Li, Christian Schilling, and Qi Zhu. ARCH-COMP22 category report: Continuous and hybrid systems with linear continuous dynamics. In *Proc. of the Int. Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*, volume 90, pp. 142–184, 2022.

Björn Lütjens, Michael Everett, and Jonathan P. How. Certified adversarial robustness for deep reinforcement learning. In *Proc. of the Conf. on Robot Learning (CoRL)*, pp. 1328–1337, 2020.

Laura Lützow and Matthias Althoff. Reachability analysis of armax models. In *Proc. of the IEEE Conf. on Decision and Control (CDC)*, pp. 7027–7034, 2023.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2018.

Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 3932–3939, 2017.

Diego Manzanas Lopez, Matthias Althoff, Marcelo Forets, Taylor T. Johnson, Tobias Ladner, and Christian Schilling. ARCH-COMP23 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In *Proc. of Int. Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23)*, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Janosch Moos, Kay Hansel, Hany Abdulsamad, Svenja Stark, Debora Clever, and Jan Peters. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 4(1):276–315, 2022.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The Int. Journal of Robotics Research*, 39:3–20, 2020.

Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. In *Proc. of the Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 2040–2042, 2018.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pp. 2817–2826, 2017.

Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279, 2019.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, volume 32, pp. 387–395, 2014.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Nicholas Socci, Daniel Lee, and H Sebastian Seung. The rectified gaussian distribution. *Advances in Neural Information Processing Systems (NeurIPS)*, 10, 1997.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2014.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Proc. of the Int. Conf. on Computer Aided Verification (CAV)*, 2020.

Fan Wu, Linyi Li, Zijian Huang, Yevgeniy Vorobeychik, Ding Zhao, and Bo Li. CROP: Certifying robust policies for reinforcement learning through functional smoothing. In *International Conference on Learning Representations*, 2022.

Huan Zhang Xiangru Zhong, Yuhao Jia. CROWN-Reach. `https://github.com/Verified-Intelligence/CROWN-Reach`, 2024.

Teng Xiao and Suhang Wang. Towards off-policy learning for ranking policies with logged feedback. *Proc. of the AAAI Conf. on Artificial Intelligence (AAAI)*, 36(8):8700–8707, 2022.

Chenxi Yang, Greg Anderson, and Swarat Chaudhuri. Certifiably Robust Reinforcement Learning through Model-Based Abstract Interpretation . In *2024 IEEE Conf. on Secure and Trustworthy Machine Learning (SaTML)*, pp. 233–251. IEEE Computer Society, 2024.

Rui Yang, Chenjia Bai, Xiaoteng Ma, Zhaoran Wang, Chongjie Zhang, and Lei Han. Rorl: Robust offline reinforcement learning via conservative smoothing. *Advances in neural information processing systems*, 35, 2022.

Zhaocong Yuan, Adam W. Hall, Siqi Zhou, Lukas Brunke, Melissa Greeff, Jacopo Panerati, and Angela P. Schoellig. Safe-Control-Gym: A unified benchmark suite for safe learning-based control and reinforcement learning in robotics. *IEEE Robotics and Automation Letters*, 7:11142–11149, 2022.

Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane Boning, and Cho-Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. In *Proc. of the 34th Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

Huan Zhang, Hongge Chen, Duane Boning, and Cho Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. In *Int. Conf. on Learning Representations (ICLR)*, 2021a.

Huan Zhang, Hongge Chen, Duane Boning, and Cho-Jui Hsieh. Robust reinforcement learning on state observations with learned optimal adversary. *arXiv preprint arXiv:2101.08452*, 2021b.

A On Set Propagation Through Neural Networks

We give additional details on the set propagation from Sec. 2.5 through the neural network in this section. In particular, the operation $\texttt{enclose}(L_k, \mathcal{H}_{k-1})$ (Prop. 1) encloses the output set of the k-th layer given the input set $\mathcal{H}_{k-1}$. If the k-th layer is linear (Def. 1), the affine map (Eq. 6) is applied:

$$\texttt{enclose}(L_k, \mathcal{H}_{k-1}) = W_k \, \mathcal{H}_{k-1} + b_k. \tag{14}$$

The output set of an activation layer is enclosed by approximating its activation function element-wise with a linear function with slope $m_k \in \mathbb{R}^{n_k}$ and approximation error $[\underline{d}_k, \overline{d}_k]$ (Koller et al., 2025):

$$\begin{aligned} \texttt{enclose}(L_k, \mathcal{H}_{k-1}) &= \text{diag}(m_k) \, \mathcal{H}_{k-1} \oplus [\underline{d}_k, \overline{d}_k], \\ m_k &= \tfrac{\sigma_k(u_{k-1}) - \sigma_k(l_{k-1})}{u_{k-1} - l_{k-1}}. \end{aligned} \tag{15}$$

Example 1. Given a toy neural network $N_\theta \colon \mathbb{R}^2 \to \mathbb{R}^2$ with $\kappa = 2$ layers, i.e., a linear layer followed by some nonlinear function. We visualize the output of this network for an input set $\mathcal{X}$ in Fig. 3(c):

$$\mathcal{X} = \left\langle \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0 & 0.5 & -0.5 \end{bmatrix} \right\rangle_Z. \tag{16}$$

Unfortunately, computing the true output set $N_\theta(\mathcal{X})$ is usually not feasible (Katz et al., 2017), and we thus need to compute an enclosure. In our toy neural network, the linear layer is given by

$$W_1 = \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix}, \quad b_1 = \mathbf{0}, \tag{17}$$

where $\phi = -\pi/4$ and thus the layer just rotates $\mathcal{X}$ by 45° counter-clockwise. Please note that this operation does not induce any outer approximation, as

$$\mathcal{H}_1 = W_1 \mathcal{X} + b_1 \approx \left\langle \begin{bmatrix} 0 \\ 0 \end{bmatrix}, 0.7 \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \right\rangle_Z \tag{18}$$

can be computed exactly using Eq. 6. However, for the nonlinear layer, we need to find a proper enclosure. By Eq. 15, this realized by approximating the activation and adding an interval error to account for the approximation error. In our example, we choose $m_k = \mathbf{1}$ for simplicity and $[\underline{d}_k, \overline{d}_k] = 0.35[-\mathbf{1}, \mathbf{1}]$. Thus,

$$\mathcal{Y} = \text{diag}(\mathbf{1})\mathcal{H}_1 \oplus 0.7 \left[\begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}, \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \right], \tag{19}$$

which gives us

$$\mathcal{Y} \approx \left\langle \begin{bmatrix} 0 \\ 0 \end{bmatrix}, 0.7 \begin{bmatrix} 1 & 1 & 0 & 0.5 & 0 \\ 0 & 1 & 1 & 0 & 0.5 \end{bmatrix} \right\rangle_Z. \tag{20}$$

Fig. 3c shows that using this enclosure, $\mathcal{Y}^* \subseteq \mathcal{Y}$ holds.

In standard training, these enclosures are not considered, so these post-hoc enclosures can become very conservative. In contrast, set-based training updates the weights considering these enclosures, and if the size of the enclosures is penalized (Fig. 4), not only is the conservativeness of the enclosure reduced, but also the true output set itself – making the networks more robust overall.

B Derivation of Set-Based Loss Functions

In this section, we motivate our choices for the set-based loss function (Prop. 3) and our set-based policy gradients (Def. 3 and 4) that are used to compute the gradient sets from a probability perspective. While maximizing the likelihood of outputs is a standard procedure to derive loss functions (Bishop & Nasrabadi, 2006, Sec. 1.2.5), lifting this theory to set-based computing has the unique advantage of integrating formal methods into the training process. To this end, we connect set-based computing and probability theory by assuming a probability distribution over the considered closed sets. For our derivations, we use a conditional posterior distribution which can be rewritten to be proportional to a likelihood function and a prior distribution (Bishop & Nasrabadi, 2006, Eq. 1.44):

$$\text{cond. posterior} \propto \text{likelihood} \cdot \text{prior}. \tag{21}$$

This reformulation is used as the posterior and is not directly obtainable, but we can assume distributions for the likelihood and the prior to obtain an estimate. In our case, the likelihood corresponds to the standard (point-based) training goal, and the prior penalizes the volume of the computed sets. As zonotopes represent these sets and thus are point-symmetric, we additionally assume that the expected value over a zonotope is its center:

$$\mathbb{E}_{z \sim \mathcal{Z}}[z] = c. \tag{22}$$

B.1 Set-Based Regression Loss

We sample random transitions $i \in [n]$ from the buffer $\mathcal{B}$ to obtain a state s_i and the corresponding actions $\tilde{a}_i \sim \tilde{\mathcal{A}}_i$. For each transition, we use Eq. 12 to obtain the critic output $\mathcal{Q}_i$ and use Eq. 3 to obtain the target y_i for each critic output $q_i \sim \mathcal{Q}_i$ using the rewards and next states stored in the buffer. To train Q_θ, we want to maximize the probability $p_\theta(q_i|y_i, s_i, \tilde{a}_i, \beta^{-1})$. Since this probability can not be computed directly, we model this probability as a conditional posterior using Eq. 21:

$$\underbrace{p_\theta(q_i|y_i, s_i, \tilde{a}_i, \beta^{-1})}_{\text{cond. posterior}} \propto \underbrace{p(y_i|q_i, \beta^{-1})}_{\text{likelihood}} \underbrace{p_\theta(q_i|s_i, \tilde{a}_i)}_{\text{prior}}. \tag{23}$$

For the prior, we assume that q_i is uniformly distributed over the interval $[l_{\mathcal{Q}_i}, u_{\mathcal{Q}_i}] \supseteq \mathcal{Q}_i \subset \mathbb{R}$, as Prop. 1 is also defined over the enclosing interval. Thus, the prior is given by

$$p_\theta(q_i|s_i, \tilde{a}_i) = \mathscr{U}(q_i|l_{\mathcal{Q}_i}, u_{\mathcal{Q}_i}) = \text{dia}(\mathcal{Q}_i)^{-1}. \tag{24}$$

As the critic learns the bootstrapped Q-iterations by regression, we assume that y_i is normally distributed with mean q_i and variance β^{-1} with the likelihood (Bishop & Nasrabadi, 2006, Eq. 1.60):

$$\begin{aligned} p(y_i|q_i, \beta^{-1}) &= \mathscr{N}(y_i|q_i, \beta^{-1}) \\ &= \sqrt{\beta/2\pi}\, \exp\big(-\beta/2\,(q_i - y_i)^2\big). \end{aligned} \tag{25}$$

Since we observe not a single q_i but an entire set $\mathcal{Q}_i$, we use the expected value $\mathbb{E}_{q_i \sim \mathcal{Q}_i}[q_i] = c_{\mathcal{Q}_i}$ Eq. 22 in the likelihood function (Bishop & Nasrabadi, 2006, Sec. 10.3). Thus, we obtain the following term to be maximized:

$$p_\theta(q_i|y_i, s_i, \tilde{a}_i, \beta^{-1}) \propto p(y_i|c_{\mathcal{Q}_i}, \beta^{-1})\, p_\theta(q_i|s_i, \tilde{a}_i) \tag{26}$$

Finally, we apply the negative logarithm and set $\beta = (\eta_Q/\epsilon)^{-1}$ to obtain our set-based loss (Prop. 3):

$$\begin{aligned} &-\ln\big(p(y_i|c_{\mathcal{Q}_i}, \beta^{-1})\, p_\theta(q_i|s_i, \tilde{a}_i)\big) \\ &\overset{25,\,24}{=} -\ln\big(\mathscr{N}(y_i|q_i, \beta^{-1})\ \text{dia}(\mathcal{Q}_i)^{-1}\big) \\ &\propto \beta/2\,(c_{\mathcal{Q}_i} - y_i)^2 + \ln\text{Dia}(G_{\mathcal{Q}_i}) \overset{\text{Prop. 3}}{\propto} L_{Reg}(y_i, \mathcal{Q}_i). \end{aligned} \tag{27}$$

We choose β that way for easier fine-tuning (Koller et al., 2025, Def. 5).

B.2 Set-Based Policy Gradient

For a state s_i, we derive the set-based policy gradient analogous to (Xiao & Wang, 2022), to maximize the probability of an action $a_i \sim \mathcal{A}_i$ being optimal given s_i – which is again not directly obtainable. Thus, we introduce a binary variable o_i indicating whether a_i is optimal and abbreviate $o_i = 1$ by o_i (Xiao & Wang, 2022, Sec. 3.1). We again model this probability as a conditional posterior over the action output using Eq. 21:

$$\underbrace{p_{\phi,\theta}(a_i|o_i, s_i, q_i, \alpha)}_{\text{cond. posterior}} \propto \underbrace{p(o_i|q_i, \alpha)}_{\text{likelihood}} \underbrace{p_{\phi,\theta}(a_i|q_i, s_i)}_{\text{prior}}. \tag{28}$$

We assume the likelihood to be exponentially distributed with $\alpha \in \mathbb{R}_+$ (Xiao & Wang, 2022, Sec. 3.2):

$$p(o_i|a_i, q_i, \alpha) = \exp(\alpha^{-1}\, q_i). \tag{29}$$

The prior in Eq. 28 is not directly computable. Thus, we model it again as a conditional posterior Eq. 21:

$$\underbrace{p_{\phi,\theta}(a_i|q_i, s_i)}_{\text{cond. posterior}} = \underbrace{p_\theta(q_i|a_i, s_i)}_{\text{likelihood}} \underbrace{p_\phi(a_i|s_i)}_{\text{prior}}, \tag{30}$$

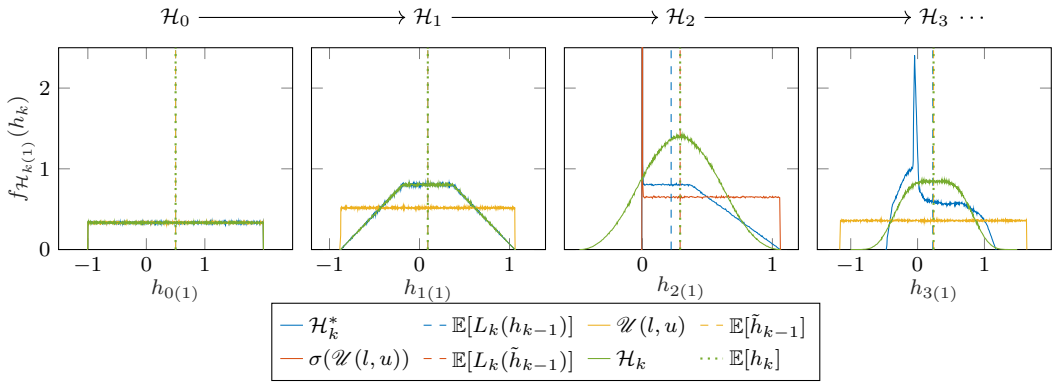

Figure 9: Propagated probability density func. $f_{\mathcal{H}_{k(1)}}(h_k)$ of zonotopes for a ReLU-neural net: True sampled density func.(blue), interval enclosure (yellow), and density of sets from Prop. 1 with $\beta_j \sim \mathscr{U}(-1,1)$ (Def. 2) (green).

where the likelihood function of q_i and the prior for a_i are uniform distributions over the enclosing intervals $[l_{\mathcal{Q}_i}, u_{\mathcal{Q}_i}] \supseteq \mathcal{Q}_i \subset \mathbb{R}$ and $[l_{\mathcal{A}_i}, u_{\mathcal{A}_i}] \supseteq \mathcal{A}_i \subset \mathbb{R}^{n_{\mathcal{A}_i}}$ analogous to Eq. 24:

$$\begin{aligned} p_\theta(q_i|a_i, s_i) &= \mathscr{U}(q_i|l_{\mathcal{Q}_i}, u_{\mathcal{Q}_i}) = \operatorname{dia}(\mathcal{Q}_i)^{-1}, \\ p_\phi(a_i|s_i) &= \mathscr{U}(a_i|l_{\mathcal{A}_i}, u_{\mathcal{A}_i}) = \prod_{j=1}^{n_{\mathcal{A}_i}} \operatorname{dia}(\mathcal{A}_i)^{-1}_{(j)}. \end{aligned} \tag{31}$$

Please note that these two probabilities correspond to the evaluation of the actor and the critic, respectively. For the likelihood function of Eq. 28, the expected value $\mathbb{E}_{q_i \sim \mathcal{Q}_i}[q_i] = c_{\mathcal{Q}_i}$ Eq. 22 is used, and taking the logarithm yields:

$$\begin{aligned} &\ln(p(o_i|c_{\mathcal{Q}_i}, \alpha)\, p_\phi(a_i|s_i)\, p_\theta(q_i|a_i, s_i)) \\ &\overset{29,31}{=} \alpha^{-1} c_{\mathcal{Q}_i} - \mathbf{1}^\top \operatorname{lnDia}(G_{\mathcal{A}_i}) - \operatorname{lnDia}(G_{\mathcal{Q}_i}). \end{aligned} \tag{32}$$

The set-based policy gradient for SA-SC (Def. 3) is derived by differentiation, where we again set the weighting factor $\alpha = (\eta_\mu/\epsilon)^{-1}$ for easier fine-tuning (Koller et al., 2025, Def. 5). Moreover, we introduce a factor $\omega \in [0,1]$ to weigh the gradients of the prior terms of $\mathcal{A}_i$ and $\mathcal{Q}_i$.

Derivation of SA-PC. For *SA-PC*, only the actor is trained using set-based training while the critic uses standard (point-based) training (① in Fig. 2). Thus, we drop the prior for the critic output q_i as this is no longer evaluated set-based and use the expected value $\mathbb{E}_{a_i \sim \mathcal{A}_i}[a_i] = c_{\mathcal{A}_i}$ for the likelihood:

$$\underbrace{p(a_i|o_i, s_i, \alpha, \phi)}_{\text{cond. posterior}} \propto \underbrace{p(o_i|s_i, c_{\mathcal{A}_i}, \alpha)}_{\text{likelihood}} \underbrace{p_\phi(a_i|s_i)}_{\text{prior}}. \tag{33}$$

Taking the logarithm while keeping our assumption on the likelihood function and the prior leads to

$$\begin{aligned} &\ln(p(o_i|s_i, c_{\mathcal{A}_i}, \alpha)\, p_\phi(a_i|s_i)) \\ &\overset{29,31}{=} \alpha^{-1} Q_\theta(s_i, c_{\mathcal{A}_i}) - \mathbf{1}^\top \operatorname{lnDia}(G_{\mathcal{A}_i}). \end{aligned} \tag{34}$$

The set-based policy gradient for SA-PC (Def. 4) is derived by differentiation and choosing α as above. This also corresponds to setting $\omega = 1$ in Def. 3.

B.3 Expectation Preserving Image Enclosure

For Eq. 26, Eq. 32 and Eq. 34, we simplify the likelihood with the expected value of the neural network output, i.e., the center. This simplification is justified with Prop. 5. In Fig. 9, we plot the probability

distributions of a set propagation and visualize the expected value for the first neuron of each layer. We compare the empirically evaluated density of a uniform input disturbance (blue) with the corresponding set-based zonotope propagation (green) across the network layers. The set propagation can be interpreted as a form of variational inference in which the output is constrained to be a zonotope. To illustrate the mean-preserving property of this single-layer variational approximation, we additionally plot the interval enclosures before activation (yellow) and the transformed intervals after activation (red). The expected value is trivially preserved for linear layers as the affine transformation for zonotopes is computed in closed form. For nonlinear layers, we observe that the expected value of the interval enclosure (red vertical line in third plot) is preserved through the enclosure (green vertical line; Prop. 1), with only small deviations to the expected value of the empirically evaluated density function (blue). Let us formally state this observation:

Proposition 5 (Tight Expectation-Preserving Set Propagation)**.** Given a neural network with ReLU-activations and an input set $\mathcal{H}_{k-1}$ with the enclosing interval $[l_{k-1}, u_{k-1}] \supseteq \mathcal{H}_{k-1}$, the expected value of the enclosure of the k-th layer is

$$\mathbb{E}_{h_k \sim \mathcal{H}_k}[h_k] = \mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[L_k(h_{k-1})],$$

with $\mathcal{H}_k = \texttt{enclose}(L_k, \mathcal{H}_{k-1})$ having minimal approximation errors.

Proof. We split cases on the type of layer L_k.

Case (1): Linear layer The expected value is preserved by linearity of the expectation:

$$\begin{aligned}\mathbb{E}_{h_k \sim \mathcal{H}_k}[h_k] &\overset{22}{=} c_k = W_k\, c_{k-1} + b_k = W_k\, \mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[h_{k-1}] + b_k \\ &= \mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[W_k\, h_{k-1} + b_k] = \mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[L_k(h_{k-1})].\end{aligned}$$

Case (2): ReLU layer Activation functions are applied element-wise, thus we consider each dimension individually; to avoid clutter, we drop the dimension index $x_{(i)}$. We distinguish between three cases: (2a) $l_{k-1}, u_{k-1} \leq 0$, (2b) $l_{k-1}, u_{k-1} \geq 0$, (2c) $l_{k-1} < 0 < u_{k-1}$. For cases (i) and (ii), ReLU is linear, thus by linearity of the expectation the expected value is preserved. For case (iii), we approximate the ReLU activation function with the affine map $m_k\, x + t_k$. We first derive a condition to ensure preserving the expected value. After that we show the slope that minimizes the approximation errors is:

$$m_k = \frac{\mathrm{ReLU}(u_{k-1}) - \mathrm{ReLU}(l_{k-1})}{u_{k-1} - l_{k-1}} = \frac{u_{k-1}}{u_{k-1} - l_{k-1}}. \tag{35}$$

The expected value is preserved if the offset t_k to satisfies the condition:

$$\begin{aligned}\mathbb{E}_{h_k \sim \mathcal{H}_k}[h_k] \overset{!}{=} \mathbb{E}[\mathrm{ReLU}(h_{k-1})] \iff \quad & c_k = \mathbb{E}[\mathrm{ReLU}(h_{k-1})] \\ \iff \quad & m_k\, c_{k-1} + t_k = \mathbb{E}[\mathrm{ReLU}(h_{k-1})] \\ \iff \quad & t_k = \mathbb{E}[\mathrm{ReLU}(h_{k-1})] - m_k\, c_{k-1},\end{aligned} \tag{36}$$

with $h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})$. Hence, we fix the offset t_k w.r.t. the slope m_k. Moreover, to find the optimal slope m_k, we compute the expected value $\mathbb{E}[\mathrm{ReLU}(h_{k-1})]$ using the probability density function $f_{\mathcal{H}_k}$ for the distribution of $\mathrm{ReLU}(\mathscr{U}(l_{k-1}, u_{k-1}))$. Therefore, we first compute the probability mass below 0 using the cumulative distribution function (Hinton & Ghahramani, 1997; Socci et al., 1997):

$$\begin{aligned}F_{\mathscr{U}(l_{k-1}, u_{k-1})}(0) &= \int_{-\infty}^{0} f_{\mathscr{U}(l_{k-1}, u_{k-1})}(h_{k-1})\, \mathrm{d}h_{k-1} \\ &= \int_{l_{k-1}}^{0} \frac{1}{u_{k-1} - l_{k-1}}\, \mathrm{d}h_{k-1} = \frac{-l_{k-1}}{u_{k-1} - l_{k-1}}.\end{aligned}$$

The probability mass is concentrated in a peak at zero using the Dirac delta $\delta(x)$ (Au & Tam, 1999). Hence, we obtain for a compactly supported function $h(x)$, with respect to the measure δ the Lebesgue integral:

$$\delta(x) = \begin{cases} \infty & x = 0, \\ 0 & \text{else}, \end{cases} \qquad \int_{-\infty}^{\infty} h(x)\, \delta(x)\, \mathrm{d}x = h(0).$$

Thus, the probability density function for the post-activation is

$$f_{\mathcal{H}_k}(h_k) = \begin{cases} \frac{1 - l_{k-1}\,\delta(h_k)}{u_{k-1} - l_{k-1}}, & 0 \leq h_k < u_{k-1}, \\ 0 & \text{otherwise.} \end{cases}$$

The resulting density is composed of the uniform distribution for the support $h_k > 0$ and the aggregated probability mass for the negative support h_{k-1} in the Dirac spike at $h_k = 0$. This follows immediately from the definition of $\mathrm{ReLU}(h_{k-1})$. Thus, the expected value of the transformed distribution is given by:

$$\begin{aligned} \mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[\mathrm{ReLU}(h_{k-1})] &= \int_{-\infty}^{\infty} \mathrm{ReLU}(h_{k-1})\, f_{\mathcal{H}_k}(\mathrm{ReLU}(h_{k-1}))\, \mathrm{d}h_{k-1} \\ &= \int_0^{u_{k-1}} h_k\, f_{\mathcal{H}_k}(h_k)\, \mathrm{d}h_k \\ &= \int_0^{u_{k-1}} h_k\, \frac{1 - l_{k-1}\,\delta(h_k)}{u_{k-1} - l_{k-1}}\, \mathrm{d}h_k \\ &= \frac{h_k^2}{2\,(u_{k-1} - l_{k-1})}\bigg|_0^{u_{k-1}} = \frac{u_{k-1}^2}{2\,(u_{k-1} - l_{k-1})}. \end{aligned}$$

Hence,

$$t_k \overset{36}{=} \frac{u_{k-1}^2}{2\,(u_{k-1} - l_{k-1})} - m_k\, c_{k-1} \overset{7}{=} \frac{1}{2}\left(\frac{u_{k-1}^2}{(u_{k-1} - l_{k-1})} - m_k\,(u_{k-1} + l_{k-1})\right). \tag{37}$$

We now find the slope m_k that minimizes the approximation errors; by satisfying Eq. 36 we ensure preserving the expected value. For concise notation, we abbreviate the approximation error at x with slope m_k by

$$d_x(m_k) \coloneqq |(m_k\, x + t_k) - \mathrm{ReLU}(x)|. \tag{38}$$

We optimize the slope m_k for minimal approximation errors:

$$\min_{m_k} \max_{x \in [l_{k-1}, u_{k-1}]} d_x(m_k).$$

With (Koller et al., 2025, Prop. 7), we know that the approximation error are located at $x \in \{l_{k-1}, 0, u_{k-1}\}$ and rewrite:

$$\min_{m_k} \max_{x \in \{l_{k-1}, 0, u_{k-1}\}} d_x(m_k).$$

From $l_{k-1} < 0 < u_{k-1}$, we can simplify

$$\begin{aligned} d_{l_{k-1}}(m_k) &= |m_k\, l_{k-1} + t_k|, \\ d_0(m_k) &= |t_k|, \\ d_{u_{k-1}}(m_k) &= |m_k\, u_{k-1} + t_k - u_{k-1}|. \end{aligned} \tag{39}$$

Moreover, we know that at least one approximation error is greater 0:

$$d_{l_{k-1}}(m_k) > 0 \vee d_0(m_k) > 0 \vee d_{u_{k-1}}(m_k) > 0.$$

Hence, the optimal slope m_k is located at an intersection of two error functions $d_{x_1}(m_k) = d_{x_2}(m_k) \geq d_{x_3}(m_k)$ with $\{x_1, x_2, x_3\} = \{l_{k-1}, 0, u_{k-1}\}$. Thus, for each intersection we find the optimal slope:

Case (2c.i): $d_{l_{k-1}}(m_k) = d_0(m_k)$

$$\begin{aligned}
& d_0(m_k) = d_{l_{k-1}}(m_k) \\
\overset{39}{\iff} \quad & |t_k| = |m_k\, l_{k-1} + t_k| \\
\iff \quad & t_k^2 = (m_k\, l_{k-1} + t_k)^2 \\
\iff \quad & 0 = (m_k\, l_{k-1})^2 + 2\, m_k\, l_{k-1}\, t_k \\
\overset{37}{\iff} \quad & 0 = m_k^2 - \frac{u_{k-1}}{u_{k-1} - l_{k-1}}\, m_k \\
\iff \quad & m_k = 0 \vee m_k = \frac{u_{k-1}}{u_{k-1} - l_{k-1}}.
\end{aligned}$$

Case (2c.ii): $d_{l_{k-1}}(m_k) = d_{u_{k-1}}(m_k)$

$$\begin{aligned}
& d_{l_{k-1}}(m_k) = d_{u_{k-1}}(m_k) \\
\overset{39}{\iff} \quad & |m_k\, l_{k-1} + t_k| = |m_k\, u_{k-1} + t_k - u_{k-1}| \\
\iff \quad & (m_k\, l_{k-1} + t_k)^2 = (m_k\, u_{k-1} + t_k - u_{k-1})^2 \\
\overset{37}{\iff} \quad & m_k\, u_{k-1}\, l_{k-1} = u_{k-1}\, \frac{u_{k-1}^2}{u_{k-1} - l_{k-1}} - u_{k-1}^2 \\
\iff \quad & m_k = \frac{u_{k-1}}{u_{k-1} - l_{k-1}}.
\end{aligned}$$

Case (2c.iii): $d_0(m_k) = d_{u_{k-1}}(m_k)$

$$\begin{aligned}
& d_0(m_k) = d_{u_{k-1}}(m_k) \\
\overset{39}{\iff} \quad & |t_k| = |m_k\, u_{k-1} + t_k - u_{k-1}| \\
\iff \quad & t_k^2 = (m_k\, u_{k-1} + t_k - u_{k-1})^2 \\
\overset{37}{\iff} \quad & -m_k^2\, l_{k-1} + m_k \left(\frac{l_{k-1}\,(2\, u_{k-1} - l_{k-1})}{u_{k-1} - l_{k-1}} \right) \\
& = \frac{u_{k-1}^2}{u_{k-1} - l_{k-1}} + u_{k-1} \\
\iff \quad & m_k = 1 \vee m_k = \frac{u_{k-1}}{u_{k-1} - l_{k-1}}.
\end{aligned}$$

In each case, we show that the proposed slope in Eq. 35 is optimal and minimizes the approximation errors $\underline{d}_k, \overline{d}_k \in \{d_{l_{k-1}}, d_0, d_{u_{k-1}}\}$.

□

C Evaluation Details

C.1 Benchmark Descriptions

Let us briefly describe the specifications of the remaining benchmarks in this section.

1D Quadrotor: The state is $s = \begin{bmatrix} z & \dot{z} \end{bmatrix}^\top$, with altitude z, velocity $\dot{z}$, and dynamics (Yuan et al., 2022):

$$\dot{s} = \begin{bmatrix} \dot{z} & \frac{a+1}{2\,m} - g \end{bmatrix}^\top, \tag{40}$$

with action space $a \in [-1, 1]$, gravity $g = 9.81$, and mass $m = 0.05$. Starting from initial states $s_0 \in [\begin{bmatrix} -4 & 0 \end{bmatrix}^\top, \begin{bmatrix} 4 & 0 \end{bmatrix}^\top]$, the Quadrotor is stabilized at $s^* = \mathbf{0}$; the reward function is $r(s_t, a_t) = -\begin{bmatrix} 1 & 0.01 \end{bmatrix} |s_{t+1} - s^*|$ for a time horizon of $3s$.

Navigation Task: We use a unicycle model with states $s = \begin{bmatrix} x & y & \theta & v \end{bmatrix}^\top$ and dynamics (Lopez et al., 2022):

$$\dot{s} = \begin{bmatrix} v\,\cos(\theta) & v\,\sin(\theta) & a_{(1)} & a_{(2)} \end{bmatrix}^\top, \tag{41}$$

with action space $a \in [-\mathbf{1}, \mathbf{1}]$. From starting point $s_0 = \begin{bmatrix} 3 & 3 & 0 & 0 \end{bmatrix}^\top$, the task is to navigate to the goal $s^* = \mathbf{0}$ without colliding with an obstacle $\mathcal{O} = [\mathbf{1}, 2 \cdot \mathbf{1}]$. The reward function is $r(s_t, a_t) = -\begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} |s_{t+1} - s^*| - c$, with $c = 1$ if $s_{t+1} \in \mathcal{O}$ and otherwise $c = 0$. We consider a time horizon of $8s$.

Inverted Pendulum: The state is $s = \begin{bmatrix} \theta & \dot{\theta} \end{bmatrix}^\top$, with angle θ, angular velocity $\dot{\theta}$, dynamics (Krasowski et al., 2023):

$$\dot{s} = \begin{bmatrix} \dot{\theta} & \frac{g}{l}\,\sin(\theta) + \frac{1}{m\,l^2}\,a \end{bmatrix}^\top, \tag{42}$$

with action space $a \in [-15, 15]$, gravity $g = 9.81$, mass $m = 1$, and length $l = 1$. The goal is to stabilize the pendulum in the upright position $s^* = \mathbf{0}$; the reward function is $r(s_t, a_t) = -\begin{bmatrix} 1 & 0.01 \end{bmatrix} |(s_{t+1} - s^*)|$ for a time horizon of $3s$.

2D Quadrotor: The state of the system is defined as $s = \begin{bmatrix} x & \dot{x} & z & \dot{z} & \theta & \dot{\theta} \end{bmatrix}^\top$, with horizontal displacement x, horizontal velocity $\dot{x}$, altitude z, vertical velocity $\dot{z}$, angle θ, angular velocity $\dot{\theta}$ and dynamics (Yuan et al., 2022):

$$\dot{s} = \begin{bmatrix} \dot{x} \\ \sin(\theta)\frac{\tilde{a}_{(1)} + \tilde{a}_{(2)}}{m} \\ \dot{z} \\ \cos(\theta)\frac{\tilde{a}_{(1)} + \tilde{a}_{(2)}}{m} - g \\ \dot{\theta} \\ \frac{l(\tilde{a}_{(2)} - \tilde{a}_{(1)})}{\sqrt{2} J_y} \end{bmatrix}, \tag{43}$$

where $\tilde{a} = (1 + \frac{1}{2}\,a)\frac{m\,g}{2}$ with action $a \in [-\mathbf{1}, \mathbf{1}] \subset \mathbb{R}^2$. The constant $g = 9.81$ defines gravity, $m = 0.027$ is the mass of the Quadrotor, $l = 0.0397$ and $J_y = 1.4 \cdot 10^{-4}$ defines the arm length of the propeller mount and the moment of inertia around the y axis. The reward function is given by $r(s_t, a_t) = -\begin{bmatrix} 1 & 0.01 & 1 & 0.01 & 0 & 0 \end{bmatrix} |s_{t+1} - s^*|$ for a time horizon of $3s$, with the goal to stabilize the Quadrotor at $s^* = \mathbf{0}$.

MuJoCo Hopper-v2: This benchmark simulates a one-legged robot hopping forward as efficiently as possible. As giving the full set of equations for this benchmark would be too exhaustive here, we refer interested readers to (Todorov et al., 2012).

C.2 Hyperparameters

We give the hyperparameters used to train the networks in our evaluation in Tab. 3.

C.3 Additional Experiments

Quadrotor 2D: We provide additional experiments on the *2D Quadrotor* and the MuJoCo *Hopper-v2* benchmark (Todorov et al., 2012). We again average the results over the last five agents in each training run and compute the mean and a 95% confidence interval across five independent random seeds. Fig. 10 compares the lower bounds $\underline{V}_\mu(s_0)$ of the different training algorithms.

Locomotion: Since, to the best of our knowledge, no existing verification toolbox can compute reachable sets for locomotion benchmarks, we are unable to provide a formal lower bound for *Hopper-v2*. However, we want to stress this is not a limitation of our training approach but rather of the subsequent verification step. Thus, to demonstrate the scalability of our approach to such benchmarks, we approximate $\underline{V}_\mu(s_0)$ using 50 trajectories under the MAD-attack (Zhang et al., 2020) and 200 trajectories perturbed with uniform random noise sampled from the ℓ_∞ ball with radius ϵ_{test}. The corresponding worst-case returns are shown in Figure 8 (MAD attack and random noise). For *Hopper-v2*, these empirical estimates provide an upper bound on the worst-case performance and demonstrate the scalability of our method. Notably, even under noise-free

Table 3: Training parameters for *PA-PC*, *SA-PC* and *SA-SC*.

Parameter	DDPG	TD3
Actor learning rate	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$
Critic learning rate	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$
Critic L_2 weight regularization λ_Q	0.01	0
Discount factor γ	0.99	0.99
Target update factor τ	0.05	0.05
Exploration noise std. deviation σ	0.1	$0.1, 0.2$
Batchsize	64	64
Buffersize	$1 \cdot 10^{6}$	$1 \cdot 10^{6}$
Episodes	2000	2000
Perturbation radius ϵ_{train}	0.1	0.1
Actor weighting factor η_μ	0.1	0.1
Critic weighting factor η_Q	0.01	0.01

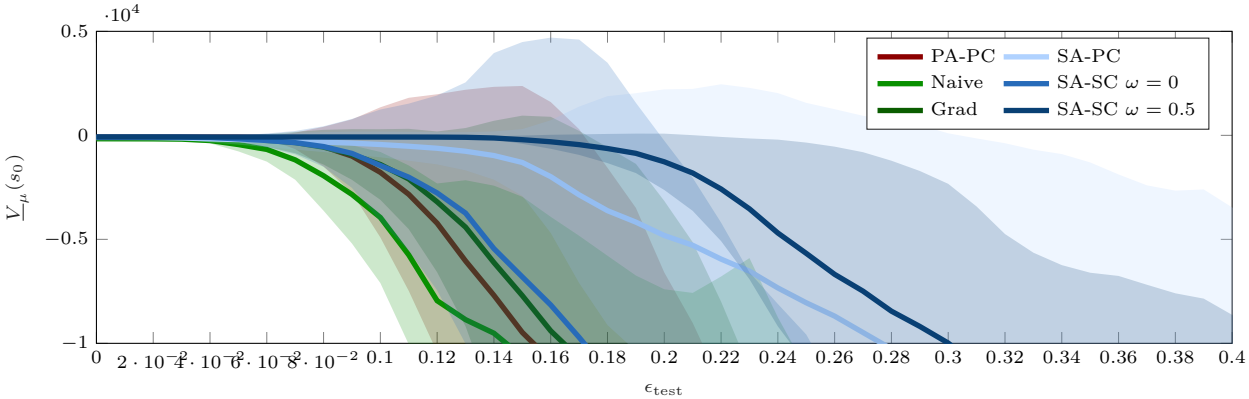

Figure 10: Comparison of $\underline{V}_\mu(s_0)$ for Quadrotor 2D.

Table 4: Training time [s/10 epochs] of *SA-PC* shows a moderate increase over baselines on Hopper-v2.

Benchmark	PA-PC	Naive	Grad	MAD	SA-PC
Hopper-v2	44.70	58.88	69.97	134.07	343.56

conditions, the worst returns of SA-PC trained agents, evaluated over randomly initialized trajectories, consistently exceed those of agents trained with a point-wise robustness criterion. We additionally provide videos illustrating agent behaviors under the uniform-random[2] and MAD[3] attacks using the first random seed, with $\epsilon_{\text{test}} = 0.1$ and $\epsilon_{\text{test}} = 0.075$. To better visualize the agents' failure modes, we disable early termination in these demonstrations. The results clearly show that the MAD attack is more effective at degrading agent performance compared to the uniform-random baseline. Notably, across both attack types, our *SA-PC* agent consistently demonstrates superior robustness and overall performance. We additionally report in Tab. 4 the run time required for ten episodes of training the locomotion benchmark Hopper-v2 for the different baselines and *SA-PC*. We report the mean runtimes over the 5 seeds, run on an Intel Core Ultra 9 185H CPU with an NVIDIA RTX 3000 Ada GPU.

Hyperparameter Discussion: Set-based reinforcement learning introduces the additional parameters η_μ and η_Q, which appear in the set-based policy gradient Def. 3 and the set-based regression loss Prop. 3. In Tab. 3, we list the hyperparameters used in our benchmarks. Notably, η_μ and η_Q are fixed in all experiments. We ease tuning these parameters by choosing β and α in Eqs. 27 and 32 as proposed by Koller et al. (2025). For a given perturbation radius ϵ_{test}, we study for *SA-PC* on the 1D Quadrotor benchmark, the effect of varying η_μ. The hyperparameter η_μ directly scales the set-based gradient of the zonotope generators. As shown in Fig. 11, increasing η_μ improves robustness: verified performance near the noise-free case $\epsilon_{\text{test}} = 0$ decreases slightly for larger η_μ, whereas verified performance at larger ϵ_{test} increases. This matches the intended trade-off between nominal (zero-perturbation) performance and robustness to larger perturbations. We therefore conclude that the choice of $\eta_\mu = 0.1$ (green) is considerably good, since it still achieves high verified rewards for large ϵ_{test}, while it preserves good rewards for the noise-free case $\epsilon_{\text{test}} = 0$. We also report the learning history for different hyperparameter settings in Fig. 12. Moderate values of η_μ lead to stable convergence, while very large η_μ slow or prevent convergence to the optimum.

We next conduct an ablation study on the hyperparameter η_Q for *SA-SC*, keeping $\eta_\mu = 0.1$ fixed. As shown in Fig. 13, large values of η_Q (e.g. $\eta_Q = 1$) make the value function highly contractive, which leads to reduced performance for small perturbation radii ϵ_{test} and to convergence difficulties, as illustrated in Fig. 14. As η_Q decreases, convergence improves and the value function captures more of the variability induced by observation noise. Consequently, the actor learns a more conservative policy, resulting in higher verified performance for larger ϵ_{test}. We find that $\eta_Q = 0.01$ (green) provides the best verified performance across all perturbation radii. Further reducing η_Q, causes verified performance to decline again.

Learning History: In Fig. 16, we present the full learning curves, which show that set-based reinforcement learning exhibits convergence behavior comparable to the adversarial baselines. For clarity, we report the rewards evaluated without observation noise during training. Additionally, we report the learning curves for the Locomotion benchmark (Hopper-v2) from Fig. 8 in Fig. 15. For this analysis, we evaluate both algorithms using the reward without observation noise. We find that, also in the Locomotion setting, our set-based method exhibits learning dynamics similar to those of the vanilla DDPG Lillicrap et al. (2016) implementation. Based on the hyperparameter analysis in Fig. 11 and 12, we also select $\eta_\mu = 0.1$ for the *Hopper-v2* benchmark.

[2]Video: `https://t1p.de/ar23e`
[3]Video: `https://t1p.de/mye6l`

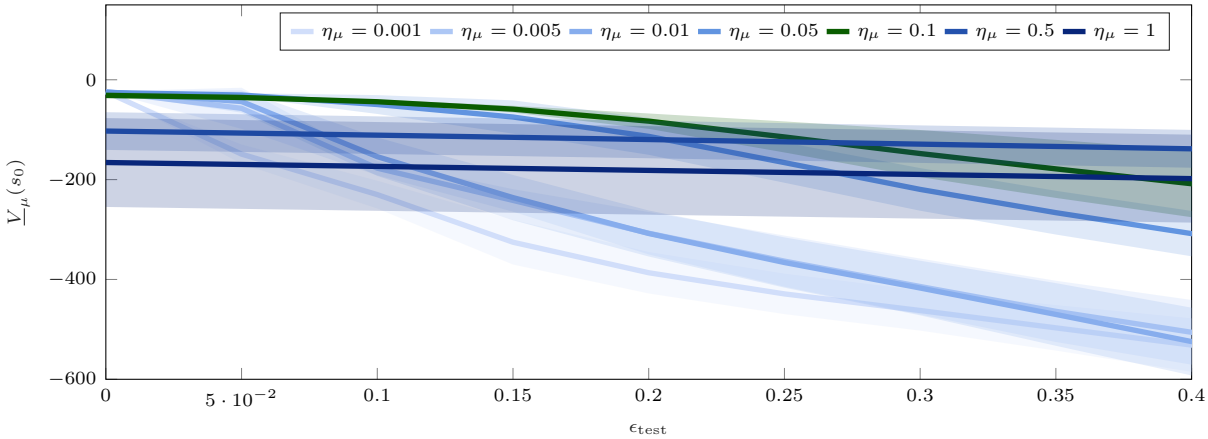

Figure 11: Tradeoff in verified performance for hyper parameter ablation η_μ.

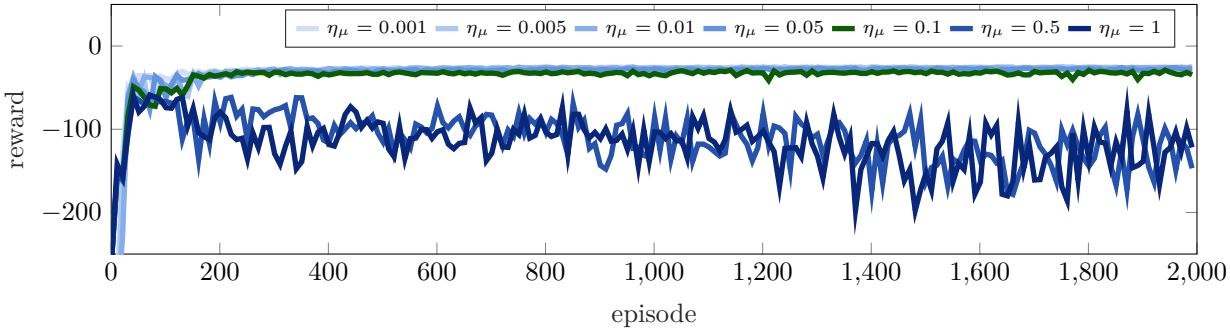

Figure 12: Learning behavior for hyper parameter ablation η_μ.

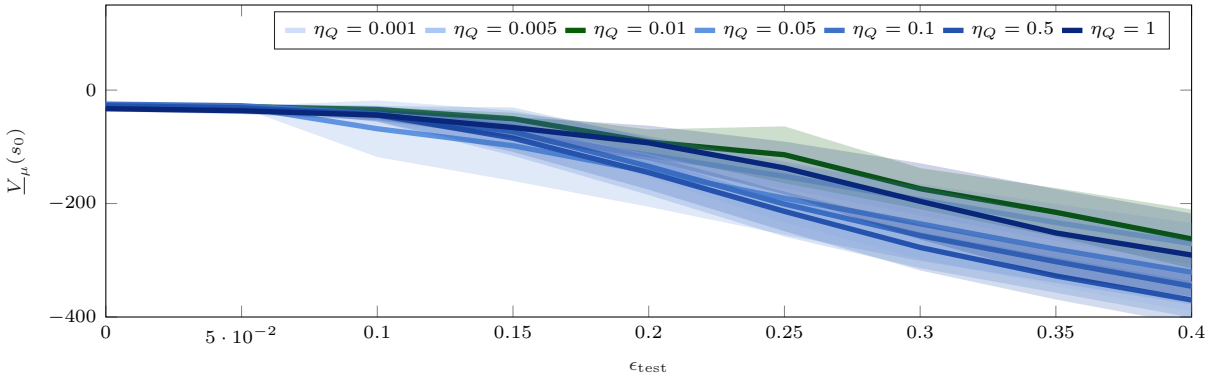

Figure 13: Hyper parameter ablation η_Q.

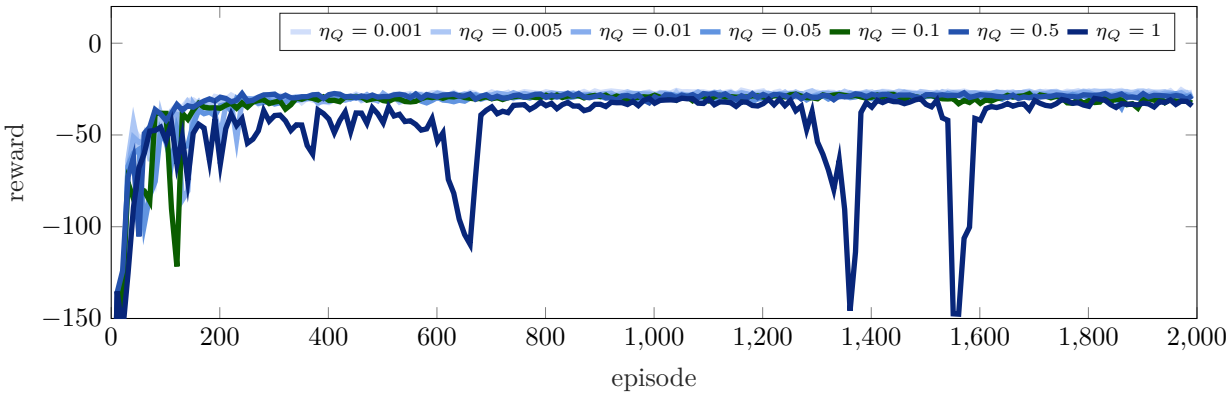

Figure 14: Learning behavior for hyper parameter ablation η_Q.

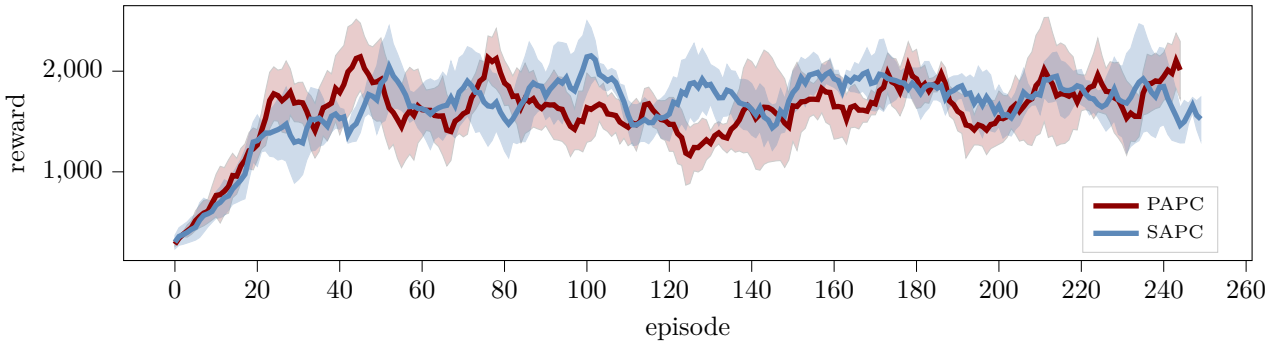

Figure 15: Full learning history for Hopper-v2.

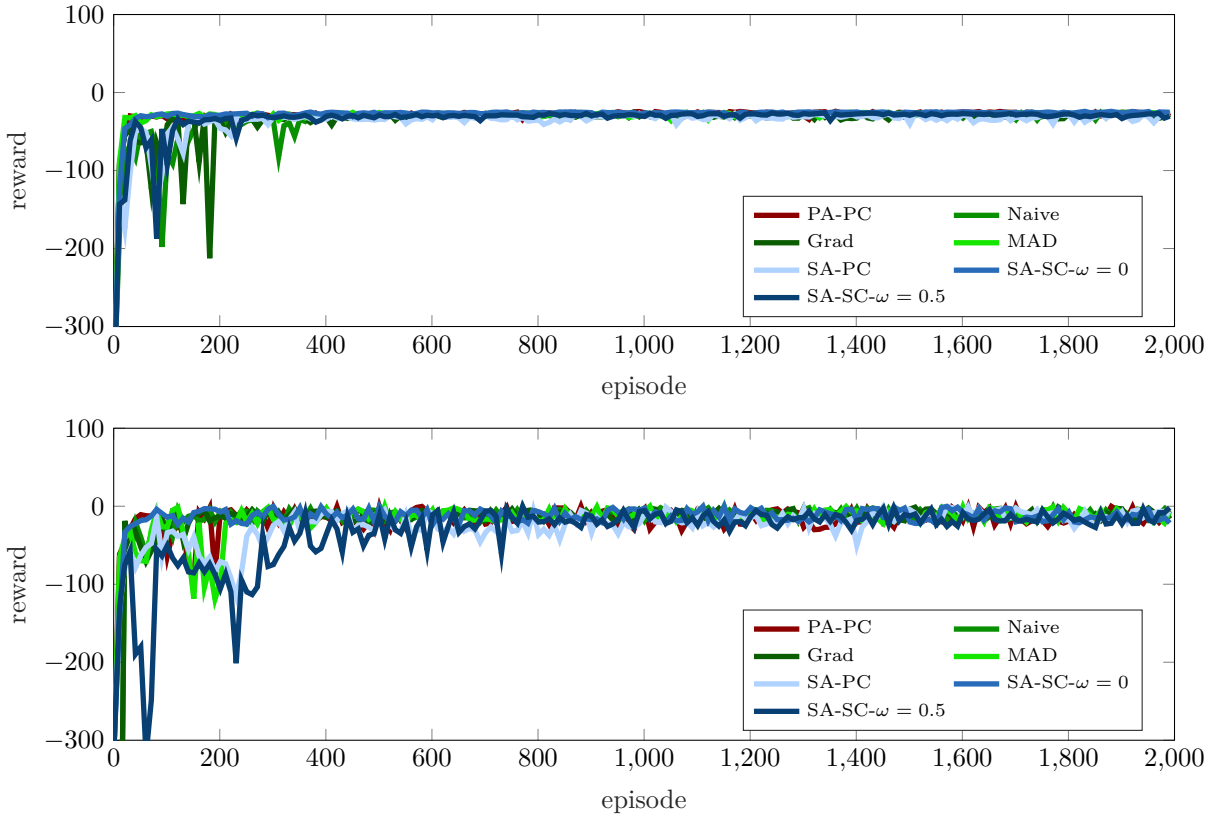

Figure 16: Full learning history for (top) 1D Quadrotor and (bottom) Inverted pendulum.