

---

# Boosting Verification of Deep Reinforcement Learning via Piece-wise Linear Decision Neural Networks

---

**Anonymous Author(s)**

Affiliation  
Address  
email

## Abstract

1 Formally verifying deep reinforcement learning (DRL) systems suffers from both  
2 inaccurate verification results and limited scalability. The major obstacle lies in the  
3 large overestimation introduced inherently during training and then transforming  
4 the inexplicable decision-making models i.e., deep neural networks (DNNs), into  
5 easy-to-verify models. In this paper, we propose an inverse *transform-then-train*  
6 approach, which first encodes a DNN into an equivalent set of efficiently and  
7 tightly verifiable linear control policies and then optimizes them via reinforcement  
8 learning. We accompany our inverse approach with a novel neural network model  
9 called *piece-wise linear decision neural networks* (PLDNNs), which are compatible  
10 with most existing DRL training algorithms with comparable performance against  
11 conventional DNNs. Our extensive experiments show that, compared to DNN-  
12 based DRL systems, PLDNN-based systems can be more efficiently and tightly  
13 verified with up to 438 times speedup and a significant reduction in overestimation.  
14 In particular, even a complex 12-dimensional DRL system is efficiently verified  
15 with up to 7 times deeper computation steps.

## 16 1 Introduction

17 Deep neural networks (DNNs) have been exhibiting appealing advantages in decision-making and  
18 control for deep reinforcement learning (DRL) systems [1–4]. Nonetheless, the complexity and  
19 inexplicability [5, 6] of DNNs render the formal verification of their hosting systems, quite often  
20 even themselves, inaccurate and unscalable. Most existing approaches [7–10] over-approximate both  
21 embedded DNNs and non-linear environment dynamics to build verifiable models, which inevitably  
22 introduces *dual* overestimation. In particular, DNN-specific overestimation is extremely unpredictable  
23 due to many factors such as the dimension of system states, the complexity of environment dynamics,  
24 and the size, weight, and activation function of a neural network. For example, the verification results  
25 may deviate significantly even if the DNNs of the same DRL system differ only in their weights (as  
26 we also observed; see Appendix A.4). Unsurprisingly, verifying high-dimensional DRL systems  
27 would only exacerbate the problems of large overestimation and limited scalability.

28 Common practice for formally verifying DRL systems is to *train and then transform* the embedded  
29 DNNs into easy-to-verify models where, for any input set, output ranges can enclose the outputs of  
30 the over-approximated DNNs [7–10]. Taylor models [11] are widely adopted due to their preservation  
31 of input-output dependencies and less overestimation (accumulated in multiple steps) than the range  
32 analysis approaches such as interval over-approximation [12, 13]. However, they are still prone to  
33 intractable overestimation as the accuracy of verification depends heavily on the weights of DNNs  
34 whose effects are difficult to quantify. Several other approaches attempt to extract approximated  
35 state-action policies, e.g., decision trees [14, 15], from DNNs via model compression [16] and  
36 distillation [17] techniques. However, no equivalence guarantee is established between DNNs and  
37 the extracted policies. Consequently, verification results are just probably approximately correct [18].

38 Inspired by recent advances [19–21] in training near-optimal policies even with reduced training  
 39 state space imposed by aggregated adjacent states, we propose a novel, inverse *transform-then-train*  
 40 approach: encoding a DNN into an equivalent set of easy-to-verify linear control policies and *then*  
 41 optimizing them by training the DNN using reinforcement learning. We accompany our inverse  
 42 approach by devising a novel neural network model called *piece-wise linear decision neural networks*  
 43 (PLDNNs), which make linear decisions on each abstract state. Unlike conventional DNNs which  
 44 build a state-action relation for each actual state, a PLDNN defines a linear relationship, called *Linear*  
 45 *Control Unit* (LCU), between actions and actual states associated with the same abstract state. To this  
 46 end, a PLDNN is essentially a set of LCUs for all abstract states. In contrast to DNNs, LCUs are  
 47 more explainable and verifiable without any over-approximation. Moreover, PLDNNs are *compatible*  
 48 with most existing DRL training algorithms as both share the same input and output layers for the  
 49 same control task.

50 We extensively assess PLDNN, along with the state-of-the-art tools, with respect to both *performance*  
 51 (in terms of cumulative rewards and system robustness) and *verifiability* (in terms of overestimation  
 52 and time cost for the reachability analysis of trained systems) on a collection of benchmarks, including  
 53 a 12-dimensional control task. Our experimental results show that, compared to the DNN-based sys-  
 54 tems, the PLDNN-based systems can be verified more precisely, with significantly less overestimation,  
 55 and more efficiently, with up to 438 times speedup, while achieving comparable performance. More-  
 56 over, compared to the state-of-the-art tools, the complex 12-dimensional control task can be trained  
 57 and verified with up to 7 times deeper computation steps, along with notable tightness improvement.

58 **Main Contributions.** Overall, we provide

- 59 1. a novel *inverse* approach for boosting the formal verification of DRL systems by learning  
 60 efficiently and directly (without over-approximation) verifiable piece-wise linear policies  
 61 with comparable performance;
- 62 2. a novel neural network model to realize the learned piece-wise linear policies, which is  
 63 compatible with most existing DRL algorithms; and
- 64 3. a prototype called LINCon, along with an extensive assessment which demonstrates its  
 65 tightness in verification results, outperformance over the state-of-the-art tools (up to 438  
 66 times speedup), and scalability (up to a 12-dimensional control task).

## 67 2 Problem Formulation and Motivation

68 A DRL system is driven by a DNN-  
 69 implemented controller  $\pi$ , which is  
 70 trained for decision-making, and a phys-  
 71 ical model defined by the ordinary dif-  
 72 ferential equations (ODEs)  $\dot{s} = f(s, a)$ ,  
 73 with  $s$  the state variables and  $a$  a con-  
 74 trol action. We have  $a = \pi(s)$  at state  
 75  $s$  and assume  $\dot{a} = 0$  during a small  
 76 time step e.g.,  $\delta$ . At the time point  
 77  $k\delta$ ,  $k \in \mathbb{N}$ , the decision network re-  
 78 ceives the current state  $s_k$  and outputs  
 79 an action  $a^{(k)} = \pi(s_k)$ . The state vari-  
 80 ables then evolve according to the phys-  
 81 ical model during the time interval  $[0, \delta]$ .  
 82 The reachable state  $s_{k+1}$  at  $\delta$  from  $s_k$  is

$$s_{k+1} = s_k + \int_0^\delta f(s, a^{(k)}) dt \quad (1)$$

**Definition 1** (Hybrid Automata [9]). A hybrid automaton is an 8-tuple  $H = \langle L, Var, Inv, F, T, G, R, I_0 \rangle$  where

- $L$  is a finite set of discrete locations;
- $Var$  is a finite set of  $n$  real-valued variables with state space  $V_c \subseteq \mathbb{R}^n$ ;
- $Inv : L \rightarrow 2^{V_c}$  is a function assigning to each location an invariant condition;
- $F : L \rightarrow (V_c \rightarrow \mathbb{R}^n)$  is a function associating each location  $l$  to a continuous dynamics  $\dot{v} = f_l(v)$ ;
- $T \subseteq L \times L$  is a set of transitions between locations;
- $G : T \rightarrow 2^{V_c}$  is a function assigning each transition  $(l_1, l_2) \in T$  a guard condition  $G(l_1, l_2) \subseteq Inv(l_1)$ ;
- $R : T \rightarrow 2^{V_c}$  is a function assigning each transition  $(l_1, l_2) \in T$  a reset  $R(l_1, l_2) \subseteq Inv(l_2)$ ; and
- $I_0 \subseteq L \times V_c$  is an initial state set.

83 which is called the successor state of  $s_k$ . Note that the system evolves continuously from  $s_k$  to  $s_{k+1}$ .  
 84 The intermediate states can be computed by substituting the time elapses for  $\delta$  in the above formula.

85 A DRL system is essentially a DNN-controlled hybrid system. Definition 1 gives a formal definition  
 86 of regular hybrid systems. The state space of a hybrid system is the Cartesian product of a set  $L$   
 87 of discrete locations and state space of  $n$  real-valued variables  $V_c$ . At each location  $l \in L$ , the  $n$

continuous variables evolve continuously according to a dynamical law  $\dot{v} = f_l(v)$ . When the guard condition on the transition between locations  $(l_i, l_2) \in T$  is triggered, the system moves to  $l_2$ , and the continuous variables are reset by  $R$ . There are two steps involved in the state transition from  $(l_i, v_i)$  to its *successor state*  $(l_{i+1}, v_{i+1})$ : first, from  $(l_i, v_i)$  to its *time successor*  $(l_i, \varphi_{f_{l_i}}(v_i, t_i))$ , and then, to  $(l_{i+1}, v_{i+1})$  that is the *transition successor* of  $(l_i, \varphi_{f_{l_i}}(v_i, t_i))$ , where  $\varphi_{f_{l_i}}$  is the solution of  $f_{l_i}$  with initial condition  $v(0) = v_i$ , mapping the initial state  $v_i$  to the state  $\varphi_{f_{l_i}}(v_i, t)$  (i.e., the reachable state at time  $t$  from  $v_i$ ). Accordingly, the state of a hybrid system can be changed in two ways [22]: (i) by a time delay that changes only the value of continuous variables according to the dynamics of the current location defined in  $F$ ; and (ii) by a discrete and instantaneous transition that changes both location and continuous variables according to the rules in  $T$ .

**Theorem 1** (Modeling DRL Systems as Hybrid Automata). *A DRL system with an environment dynamics  $f$ , decision network  $\pi$ , time step size  $\delta$ , and initial state set  $S_0$ , can be modeled as the following hybrid automaton:*

- *Var: state variable  $s$ , action  $a$ , and clock variable  $t_c$*      •  $I_0: \{(l_0, (s \in S_0, a = 0, t_c = \delta))\}$
- $L: \{l_0\}$      •  $Inv: Inv(l_0) = \{s \in S, t_c \leq \delta\}$      •  $F: F(l_0) = \{\dot{s} = f(s, a), \dot{a} = 0, \dot{t}_c = 1\}$
- $T: \{(l_0, l_0)\}$      •  $G: G(l_0, l_0) = \{t_c = \delta\}$      •  $R: R(l_0, l_0) = \{t_c = 0, a = \pi(s)\}$

Figure 1 depicts the hybrid automaton defined in Theorem 1. There exists only one location  $l_0$ . The invariants in  $Inv$  claim that any state belongs to the state space  $S$  and the clock variable  $t_c$  is less than or equal to the time step size  $\delta$ . The flow in  $F$  defines the dynamics of the system. The only transition is triggered when  $t_c = \delta$ , updating the action  $a$  and resetting  $t_c$  as defined in  $R$ . The continuous change happens during the time interval  $[i\delta, (i+1)\delta]$ , with  $i \in \mathbb{N}$ , and the discrete change of actions occurs at each  $\delta$ .

Unfortunately, the hybrid automaton of a DRL system cannot be verified by using existing hybrid automata model checkers such as Flow\* [23], Ariadne [24], and CORA [25]. The reason is that the action  $a$  in  $R$  depends on the uninterpretable DNN  $\pi$  by  $a := \pi(s)$ , and  $\dot{s} = f(s, a)$  can not be expressed in a known closed-form which is required by regular hybrid automata supported by these tools [26]. Hence, almost all reachability-based verification methods for DRL systems such as Polar [10], Sherlock [7], and ReachNN [27] inevitably over-approximate  $\pi$  using a Taylor model, at the cost of large overestimation and time overhead.

### 3 Piece-Wise Linear Control Policies

To bypass the crux of over-approximating DNNs, we devise a novel, alternative neural network model which essentially realizes a set of linear control policies. Our approach bases on the common assumption that there exists a near-optimal linear control policy for every small region of the entire state space [28, 29, 21, 30]. Our objective is then to discretize the state space  $S$  of a DRL system and to train a linear control policy for each discretized region. Specifically, given an  $n$ -dimensional DRL system with  $m$ -dimensional control input, we train a DNN which implements a linear control function  $a_j = b^j + c_1^j x_1 + c_2^j x_2 + \dots + c_n^j x_n$  for each control dimension  $1 \leq j \leq m$  and each discretized region.

#### 3.1 Abstracting State Space via Abstract Interpretation

Abstract Interpretation [31] is an effective technique for scaling up formal verification of complex systems or programs by reducing the system space while preserving the soundness of verification results. For instance, an infinite state space  $[-2, 0] \times [0, 2]$  can be abstracted to be an abstract state represented as  $(-2, 0, 0, 2)$ , when all the states in  $[-2, 0] \times [0, 2]$  share a same property. In general, given a system state space  $S$ , we denote  $S_\phi$  as a finite set of abstract states (each abstract state represents a possibly infinite set of actual system states in  $S$ ). Let  $\phi : S \rightarrow S_\phi$  be an abstraction function that maps each actual state  $s$  in  $S$  to the corresponding abstract state in  $S_\phi$ , and  $\phi^{-1} : S_\phi \rightarrow 2^S$  be the inverse concretization function such that  $\phi^{-1}(s_\phi) = \{s | s \in S, \phi(s) = s_\phi\}$ .

For state space abstraction, we choose the very primitive but effective abstraction approach which abstracts actual system states as intervals. It is known as *interval abstract domain* and has been

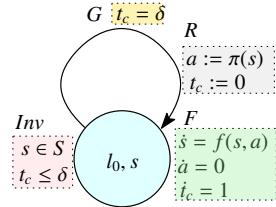


Figure 1: Hybrid automaton for a DRL system.

139 well studied for system [32] and program verification [33] and even the approximation of neural  
 140 networks [34]. Specifically, let  $L_i$  and  $U_i$  be the lower and upper bounds for the  $i$ -th dimension value  
 141 of  $S$ . We first define the abstraction granularity as an  $n$ -dimensional vector  $\gamma = (d_1, d_2, \dots, d_n)$ . Then  
 142 the  $i$ -th dimension will be divided evenly into  $(U_i - L_i)/d_i$  intervals which means each abstract state  
 143 can be represented as a  $2n$ -dimensional vector  $(l_1, u_1, \dots, l_n, u_n)$ .

144 **Definition 2** (Interval-Based Abstraction Function). *Given an  $n$ -dimensional continuous state space  
 145  $S$  and an abstract state space  $S_\phi$  which discretizes  $S$  based on abstraction granularity  $\gamma$ ,  $\phi : S \rightarrow S_\phi$   
 146 is called an interval-based abstraction function such that, for every actual state  $s = (x_1, \dots, x_n) \in S$   
 147 and abstract state  $s_\phi = (l_1, u_1, \dots, l_n, u_n) \in S_\phi$ , we have  $\phi(s) = s_\phi$  if and only if  $l_i \leq x_i < u_i$  holds for  
 148 each dimension  $1 \leq i \leq n$ .*

149 **Example 1** (Running Example). *Consider a 2-dimensional system in [35] with state space  $[-2, 2] \times [-2, 2]$ .  
 150 The dynamics  $f$  is defined by following ordinary differential equations (ODE) i.e.,  $\dot{x}_1 = x_2 - x_1^3$   
 151 and  $\dot{x}_2 = a$ . The sign  $a$  means the control action. The objective is to train a DNN for determining  
 152 action  $a$  based on  $(x_1, x_2)$  so that the agent can move from the initial region  $x_1 \in [0.7, 0.9]$  and  
 153  $x_2 \in [0.7, 0.9]$  to the goal region  $x_1 \in [-0.3, 0.1]$  and  $x_2 \in [-0.35, 0.05]$  as soon as possible.*

154 Suppose that the abstraction granularity is  $\gamma = (2, 2)$ . The continuous state space  $[-2, 2] \times [-2, 2]$   
 155 is then partitioned into four regions, corresponding to four abstract states represented by  $S_\phi =$   
 156  $\{(-2, 0, -2, 0), (-2, 0, 0, 2), (0, 2, -2, 0), (0, 2, 0, 2)\}$ , respectively.

### 157 3.2 Piece-Wise Linear Decision Neural Networks

158 We devise an alternative DNN model called *piece-wise linear decision neural networks* (PLDNNs).  
 159 Unlike conventional DNNs, a PLDNN contains an *abstraction layer* between the input layer and the  
 160 first hidden layer. The abstraction layer is used to convert an actual system state into its corresponding  
 161 abstract state. Then the output of a PLDNN is the control action that is the *dot product* result of state  
 162 variables of the actual state and the linear coefficients determined by the corresponding abstract state.

163 Figure 2 exemplifies the architecture of the  
 164 PLDNN  $\pi$  for a two-dimensional DRL system.  
 165 The decision-making of  $\pi$  is based on a coeffi-  
 166 cient network  $\pi_c$  that outputs the linear coeffi-  
 167 cients. The second layer of  $\pi_c$  is the inserted  
 168 abstraction layer which consists of the blue  
 169 neurons and the red neurons. The output layer of  $\pi_c$   
 170 contains  $n+1$  neurons that output the  $n+1$  linear  
 171 coefficients depicted as purple neurons. As for  
 172 the weights setting between the input layer and  
 173 the abstraction layer, the weights of the connec-  
 174 tions between the  $i$ -th neuron in the input layer  
 175 and the  $(2i-1)$ -th and  $2i$ -th neurons in the ab-  
 176 straction layer are set to 1 which are represented  
 177 by blue lines and red lines, respectively. While the weights of other connections denoted by the black  
 178 dashed lines are set to 0. Under this setting of weights,  
 179 the inputs to both  $(2i-1)$ -th and  $2i$ -th neurons  
 180 in the abstraction layer are  $x_i$ . Moreover, the activation function of the  $(2i-1)$ -th neuron in the  
 181 abstraction layer is set to  $\phi_u$  with the responsibility of computing the upper bound  $u_i$ , and that of the  
 182  $2i$ -th neuron is set to  $\phi_l$  for calculating the lower bound  $l_i$ . Specifically, for a continuous state space  
 183 partitioned by the abstraction granularity  $\gamma = (d_1, \dots, d_n)$ , the activation functions for the  $(2i-1)$ -th  
 and  $2i$ -th neurons in the abstraction layer can be formulated as follows:

$$\phi_l^i(x_i) = L_i + \lfloor \frac{(x_i - L_i)}{d_i} \rfloor d_i \quad \phi_u^i(x_i) = L_i + \lfloor \frac{(x_i - L_i)}{d_i} \rfloor d_i + d_i$$

184 With the above activation functions, the abstraction layer can output the same abstract state  $s_\phi =$   
 185  $(l_1, u_1, \dots, l_n, u_n)$  for  $\forall s \in \phi^{-1}(s_\phi)$ . The abstract state  $s_\phi$  is then propagated to the fully connected  
 186 layers of  $\pi_c$  to generate the linear control coefficients  $(b, c_1, c_2)$  denoted by  $\pi_c(s)$ .

187 To obtain the final output of action  $a$ , an additional dot product operation between  $\pi_c(s)$  and  $[1, s]$   
 188 is performed with the result of the operation as the control action  $a$  where  $[\cdot, \cdot]$  is the concatenation  
 189 operation. For multiple dimensional control action  $a = (a_1, \dots, a_m)$ , we only need to modify the

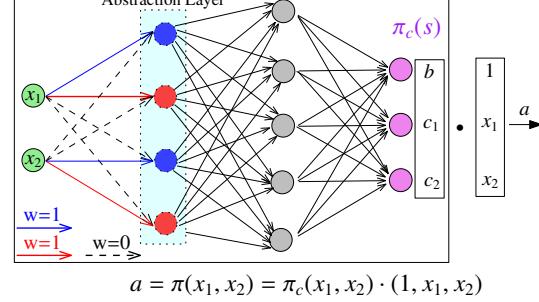


Figure 2: The arch. of the PLDNN for Example 1.

---

**Algorithm 1:** The Training Procedure based on the DDPG algorithm

---

```

1 Input: State space  $S$ , abstraction granularity  $\gamma$ . Output: A PLDNN  $\pi$ 
2  $\phi \leftarrow$  discretize  $S$  according to  $\gamma$ ; // obtain abstraction function, Sec3.1
3 Initialize actor network  $\pi$  as a PLDNN by encoding  $\phi$  into the coefficient network  $\pi_c$ ; // Sec3.2
4 Initialize critic network  $Q$ , target networks  $\pi' \leftarrow \pi$ ,  $Q' \leftarrow Q$ ;
5 DDPG( $\pi, Q, \pi', Q'$ ); // train  $\pi$  based on DDPG algorithm
6 return  $\pi$ 

```

---

190 output dimension of  $\pi_c$  to  $m(n + 1)$ , such that each  $n + 1$  neurons output the linear coefficients of one  
 191 dimension of  $a$ . More specifically, we can obtain  $a_j$  as follows:

$$a_j = \pi(s)_j = \pi_c(s)[(n + 1)(j - 1) : (n + 1)j] \cdot [1, s], 1 \leq j \leq m.$$

192 where  $vector[start : end]$  denotes the slicing operation that extracts the elements of  $vector$  from  
 193 index  $start$  up to but not including index  $end$ .  
 194 With the additional abstraction layer that can output an identical vector into the fully connected layers  
 195 of  $\pi_c$  for  $\forall s \in \phi^{-1}(s_\phi)$ , we can ensure that  $\pi_c$  always produces the same coefficients for all actual  
 196 states located in the same abstract state. Consequently, we can extract a piecewise linear decision  
 197 function with this structure of  $\pi$  on each abstract state.

198 **3.3 The Training Procedure**

199 Training a PLDNN can be achieved by extending existing  
 200 deterministic policy gradient algorithms such as Deep  
 201 Deterministic Policy Gradient (DDPG) [36] and Twin De-  
 202 layed Deep Deterministic Policy Gradient [37] since the  
 203 modifications made stay inside neural networks and are  
 204 invisible to the DRL algorithms. The pseudo code of  
 205 the training procedure is given in Algorithm 1, where we  
 206 take the DDPG algorithm as an example. The training  
 207 procedure starts with defining the abstraction function  $\phi$   
 208 according to  $\gamma$  (Line 2), initializing PLDNN with an ab-  
 209 straction layer based on  $\phi$  (Line 3), and, following [36],  
 210 initializing the critic network and the two target networks  
 211 (Line 4). The procedure then invokes the DDPG algorithm with the networks as arguments (Line  
 212 5) since the PLDNN has the same input and output as the actor network implemented by DNN.  
 213 During this procedure, we freeze the parameters between the input and the abstraction layers of  $\pi$ .  
 214 The parameters in the fully connected layers are trained based on backpropagation [38] and gradient  
 215 descent optimization [39].

216 **4 Equivalent Policy Extraction and Verification**

217 After training, we can extract  $|S_\phi|$  LCUs based on the learned coefficients of linear control policies  
 218 for the abstract states in  $S_\phi$ . Specifically, we choose an actual state for each abstract state and feed it  
 219 to a PLDNN to obtain the coefficients for the abstract state. For instance, we can feed  $(-1, -1)$  to  
 220 the PLDNN in Example 1 and obtain the coefficients  $(-0.16610657, -1.7437580, -1.8227874)$  of  
 221 the linear control policy for the region  $[-2, 0] \times [-2, 0]$ . Figure 3 shows the LCUs extracted from a  
 222 trained PLDNN in Example 1. They are depicted by planes with different colors in Figure 3. These  
 223 four planes denote the following linear control functions:

$$\begin{aligned} \pi(x_1, x_2) &= -0.16610657 - 1.7437580x_1 - 1.8227874x_2, & x_1 \in [-2, 0], \quad x_2 \in [-2, 0] & \text{(LCU}_1\text{)} \\ \pi(x_1, x_2) &= -0.20400035 - 1.8006037x_1 - 1.8679885x_2, & x_1 \in [-2, 0], \quad x_2 \in [0, 2] & \text{(LCU}_2\text{)} \\ \pi(x_1, x_2) &= -0.27547930 - 1.8884722x_1 - 1.9342268x_2, & x_1 \in [0, 2], \quad x_2 \in [-2, 0] & \text{(LCU}_3\text{)} \\ \pi(x_1, x_2) &= -0.29549897 - 1.9022338x_1 - 1.9436346x_2, & x_1 \in [0, 2], \quad x_2 \in [0, 2] & \text{(LCU}_4\text{)} \end{aligned}$$

224 The underlying  $x_1 \times x_2$  plane in Figure 3 is the projection of the four LCUs. Under the control of  
 225 these four LCUs, the agent can reach the goal region (orange box). The two sequences of purple  
 226 boxes represent the range of reachable states from corresponding initial regions to the goal region.

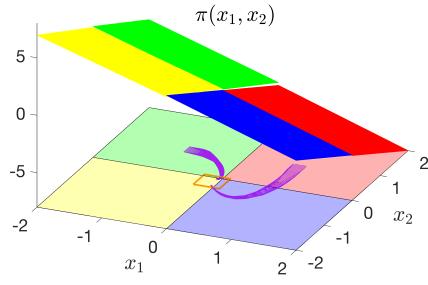


Figure 3: The LCUs extracted from a trained PLDNN for Example 1.

With exacted LCUs from a PLDNN, we can build a verifiable hybrid automaton for the system by substituting equivalently the neural networks using corresponding LCUs. Theorem 2 formulates the hybrid automaton after a decision network  $\pi$  is substituted by LCUs. The differences from Theorem 1 include the definitions of transitions  $T$ , guard condition  $G$  and reset formula  $R$ . For the pldnn controlled systems, We use  $|S_\phi|$  transitions each of which contains a guard condition and a reset formula to update the action  $a$  to  $\pi_c(s) \cdot [1, s]$  at each  $\delta$ . Notice that,  $\pi_c(s)$  is a determined vector since  $\forall s \in \phi^{-1}(s_\phi^i)$ , the outputs of  $\pi_c$  are the same according to the dedicated structure of  $\pi_c$ . Thus, the reset formula for  $a$  is simplified to an affine mapping.

**Theorem 2.** Given a DRL system with environment dynamics  $f$ , PLDNN  $\pi$ , time step size  $\delta$  and initial state set  $S_0$ , it can be equivalently modeled as a hybrid automaton as follows:

- Var: state variable  $s$ , action  $a$ , clock variable  $t_c$
- $I_0: \{(l_0, (s \in S_0, a = 0, t_c = \delta))\}$
- $L: \{l_0\}$
- Inv:  $Inv(l_0) = \{s \in S, t_c \leq \delta\}$
- $F: F(l_0) = \{\dot{s} = f(s, a), \dot{a} = 0, \dot{t}_c = 1\}$
- $T: \{(l_0, l_0), \dots, (l_0, l_0)\}$  where  $|T| = |S_\phi|$
- $G: G[T[i]] = \{t_c = \delta, s \in \phi^{-1}(s_\phi^i)\}$  where  $0 \leq i < |T| \wedge s_\phi^i \in S_\phi$
- $R: R(T[i]) = \{t_c = 0, a = \pi_c(s) \cdot [1, s]\}$  where  $0 \leq i < |T| \wedge s \in \phi^{-1}(s_\phi^i)$

Our soundness proofs of both Theorem 1 and Theorem 2 are given in Appendix A.5. Based on Theorem 2, we can build a hybrid automaton for the DRL system in Example 1. Assuming the trained system has four linear control units as shown in Formulas (LCU<sub>1</sub>-LCU<sub>4</sub>) and  $\delta = 0.2$ , we construct the corresponding hybrid automaton as depicted in Figure 4. The four transitions in the automaton correspond to the four LCUs, respectively. The guard of each transition represents the condition of triggering the corresponding policy.

Thanks to the linearity of control policies in  $R$ , a hybrid automaton built for a PLDNN-controlled system can be efficiently verified by state-of-the-art tools. For instance, Flow\* [23] is a representative tool for the reachability analysis of hybrid systems. In this paper we are focused on the verification of both goal-reach and reach-avoid properties. The former means that given a set of initial states, a system must eventually reach the goal region from any initial state. The latter means that the system never enters unsafe regions within a specific time horizon. Both properties can be verified via reachability analysis.

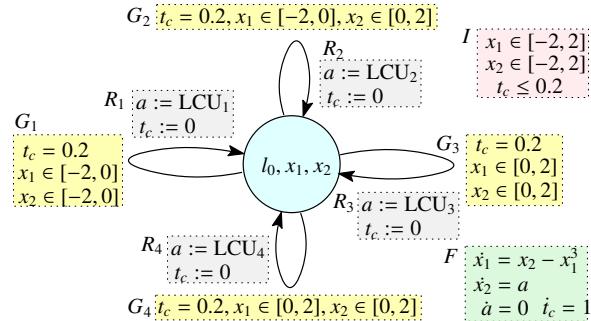


Figure 4: The hybrid automaton of the DRL system with the trained piece-wise linear controllers in Figure 3.

## 5 Experiments

We prototype our approach into a tool called LinCon, with DDPG as the backend DRL algorithm and Flow\* as the verification engine. We extensively assess it, along with the state-of-the-art tools. Our goal is to show, for the PLDNN-based training, (i) the reduction in the number of partitions with comparable cumulative rewards, robustness, and time overhead with respect to conventional DNN-based training; (ii) its high verification performance including the tightness of over-approximation sets and the efficiency of verification; and (iii) its scalability for large-sized neural networks and systems with complex dynamics and high-dimensional state space.

**Experimental Setup.** All experiments were conducted on a workstation equipped with a 32-core AMD Ryzen Threadripper CPU @ 3.6GHz and 256GB RAM, running Ubuntu 22.04.

**Benchmarks.** We choose eight benchmarks, including six regular benchmarks from Verisig 2.0 [9] (B1-B5 and Tora) and two complex benchmarks (CartPole with extreme complex dynamics from OpenAI Gym [40] and quadrotor (QUAD) with 12-dimensional state space and 3-dimensional action space from [10]). For fair comparisons, we use the same training configuration and guarantee that all trained systems reach the specified reward threshold. See Appendix A.1 for the detailed setting.

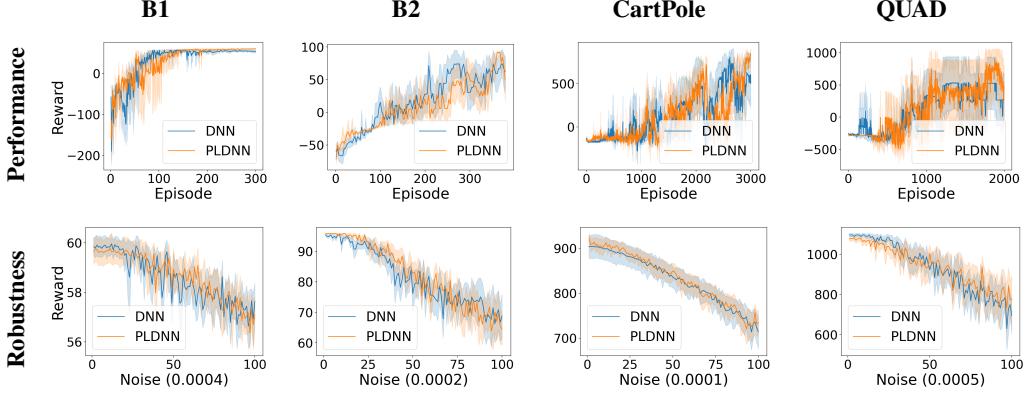


Figure 5: Performance and robustness comparison between PLDNNs and DNNs. The number in the parentheses is the base of  $\sigma$ , e.g., when the abscissa is 50 in B1, we have  $\sigma = 50 \times 0.0004 = 0.02$ .

### 278 5.1 Performance Evaluation

279 We assess the performance of PLDNNs, together with the conventional DNNs, in terms of cumulative  
 280 reward, robustness, and training time under the same training configuration. We also measure the  
 281 number of abstracted states required for training linear control policies and constant policies [21].  
 282 Due to space limitations, we present the experimental results only for B1, B2, and two complex cases  
 283 (i.e., CartPole and QUAD). The associated conclusions (from Table 1 and Figure 5) also apply to the  
 284 other four cases; see Appendix A.2 for the detailed experimental results.

285 **Cumulative Reward.** Figure 5 plots the  
 286 system cumulative reward (the average of  
 287 five trials) during the training process. The  
 288 solid lines and shadows refer to the aver-  
 289 age reward and 95% confidence interval,  
 290 respectively. Apparently, the trends of ac-  
 291 cummulative rewards by PLDNNs and DNNs  
 292 are comparable. Despite slightly more time required for training with PLDNN due to its additional  
 293 conversion from actual state to abstract state (see Table 1, ‘‘Training Time’’), we can, however, obtain  
 294 a significant advance in the efficiency and tightness of verification (see Section 5.2).

295 **Robustness.** We also evaluate the robustness of PLDNNs as training the linear controller on each  
 296 partition may lead to discontinuity of control decisions in the boundaries of partitions. For a  
 297 current state  $s = (x_1, \dots, x_n)$ , we add a Gaussian noise  $X_1, \dots, X_n$  to  $s$  and obtain a perturbed state  
 298  $s' = (x_1 + X_1, \dots, x_n + X_n)$  for calculating the control action, where  $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$  with  $1 \leq i \leq n$   
 299 and  $\mu_i = 0$ . For each benchmark, we train 10 different policies and evaluate their robustness under  
 300 100 different perturbation levels to obtain the average and 95% confidence interval of the cumulative  
 301 reward. Figure 5 depicts the reward trend with the increasing perturbation level. As  $\sigma$  increases, the  
 302 decline ratio of the system with PLDNNs is comparable to that with DNNs, which implies that both  
 303 achieve similar robustness.

304 **Reduction in the Number of Partitions.** We measure the effect of reducing the number of partitions  
 305 by utilizing a linear policy, instead of the constant action on each partition. In both cases, we start  
 306 training from a coarse-grained abstraction granularity (with only one partition) and gradually refine  
 307 the abstraction granularity until the preset reward threshold is satisfied. As shown in Table 1, linear  
 308 policies significantly reduce the number of partitions required for reaching the reward threshold,  
 309 which benefits the verification efficiency as we will see next.

### 310 5.2 Verification Efficiency

311 We evaluate the verification efficiency and tightness for the PLDNN-based and DNN-based DRL  
 312 systems, respectively. Regarding tightness, we choose two state-of-the-art tools, i.e., Polar [10]  
 313 and Verisig 2.0 [9], for the reachability analysis of DNN-controlled systems. We do not consider  
 314 ReachNN\* which both Polar and Verisig 2.0 have been demonstrated to outperform [9, 10]. For  
 315 efficiency, we employ Flow\* to perform reachability analysis, which is used by both Polar and Verisig  
 316 2.0 as the backend reachability analysis tool.

Table 2: Verification results and time in seconds.

Task	Dim	Network	LinCon		Polar			Verisig 2.0				
			1 Core	V.R.	1 Core	Impr.	V.R.	1 Core	Impr.	20 Cores	Impr.	V.R.
<b>B1</b>	2	Tanh <sub>2×20</sub>	2.31	✓	17	7.4×	✓	45	19.5×	38	16.5×	✓
		Tanh <sub>3×100</sub>	2.28	✓	125	54.8×	✓	413	181.1×	123	53.9×	✓
		ReLU <sub>2×20</sub>	2.11	✓	3	1.4×	✓	—	—	—	—	X <sup>c</sup>
		ReLU <sub>3×100</sub>	2.59	✓	—	—	X <sup>b</sup>	—	—	—	—	—
<b>B2</b>	2	Tanh <sub>2×20</sub>	0.57	✓	5	8.8×	✓	5	8.8×	4	7.0×	X <sup>a</sup>
		Tanh <sub>3×100</sub>	0.56	✓	—	—	X <sup>b</sup>	—	—	—	—	X <sup>b</sup>
		ReLU <sub>2×20</sub>	0.64	✓	3	4.7×	✓	—	—	—	—	X <sup>c</sup>
		ReLU <sub>3×100</sub>	0.60	✓	—	—	X <sup>b</sup>	—	—	—	—	—
<b>B3</b>	2	Tanh <sub>2×20</sub>	2.69	✓	18	6.7×	✓	36	13.4×	28	10.4×	✓
		Tanh <sub>3×100</sub>	3.57	✓	91	25.5×	✓	357	100.0×	88	24.6×	✓
		ReLU <sub>2×20</sub>	3.05	✓	8	2.6×	✓	—	—	—	—	X <sup>c</sup>
		ReLU <sub>3×100</sub>	2.92	✓	14	4.8×	✓	—	—	—	—	X <sup>c</sup>
<b>B4</b>	3	Tanh <sub>2×20</sub>	1.44	✓	5	3.5×	✓	7	4.9×	5	3.5×	✓
		Tanh <sub>3×100</sub>	1.45	✓	27	18.6×	✓	114	78.6×	31	21.4×	✓
		ReLU <sub>2×20</sub>	1.43	✓	2	1.4×	✓	—	—	—	—	X <sup>c</sup>
		ReLU <sub>3×100</sub>	1.43	✓	5	3.5%	✓	—	—	—	—	X <sup>c</sup>
<b>B5</b>	3	Tanh <sub>3×100</sub>	3.24	✓	38	11.7×	✓	157	48.5×	44	13.4×	✓
		Tanh <sub>4×200</sub>	3.29	✓	157	47.7×	✓	1443	438.6×	191	58.1×	✓
		ReLU <sub>3×100</sub>	3.28	✓	7	2.1×	✓	—	—	—	—	X <sup>c</sup>
		ReLU <sub>4×200</sub>	3.29	✓	49	14.9%	✓	—	—	—	—	X <sup>c</sup>
<b>Tora</b>	4	Tanh <sub>3×20</sub>	1.57	✓	45	28.7×	✓	69	43.9×	46	29.3×	✓
		Tanh <sub>4×100</sub>	1.75	✓	—	—	X <sup>b</sup>	—	—	—	—	X <sup>b</sup>
		ReLU <sub>3×20</sub>	1.58	✓	30	19.0%	✓	—	—	—	—	X <sup>c</sup>
		ReLU <sub>4×100</sub>	1.62	✓	53	32.7%	✓	—	—	—	—	X <sup>b</sup>
CartPole	4	Tanh <sub>3×64</sub>	151	✓	—	—	X <sup>b</sup>	—	—	—	—	X <sup>b</sup>
QUAD	12	Tanh <sub>3×64</sub>	1054	✓	—	—	X <sup>b</sup>	—	—	—	—	X <sup>b</sup>

**Remarks.** **Impr.:** time speedup of LinCon compared to Verisig or Polar (Verisig or Polar/LinCon).  
**Tanh/ReLU<sub>n×k</sub>:** a DNN with the activation function Tanh/ReLU,  $n$  hidden layers, and  $k$  neurons per hidden layer. **VR:** verification result. ✓: the reachability problem is successfully verified.  
**X<sup>type</sup>:** the reachability problem cannot be verified due to type: (a) large over-approximation error, (b) the calculation did not finish, (c) not applicable. —: no data available due to X<sup>b</sup> or X<sup>c</sup>.

317 **Efficiency.** Table 2 presents the comparison results for the verification efficiency. For each regular  
318 case, we choose four different network configurations: two smaller networks (e.g., Tanh<sub>2×20</sub>) from [10]  
319 and two larger networks (e.g., Tanh<sub>3×100</sub>). Our approach LinCon can handle all 26 instances including  
320 the two complex instances, while Polar succeeds only in 20 cases. Verisig 2.0 is not applicable to  
321 ReLU networks (marked by X<sup>c</sup>) and succeeds only in 9 instances. Overall, LinCon outperforms both  
322 Polar (up to 47.7× speedup) and Verisig 2.0 (up to 438.6× speedup). In particular, LinCon achieves  
323 even up to 58.1× speedup compared to Verisig 2.0 accelerated by 20-core parallelization. For our  
324 approach LinCon, the only time overhead for encoding networks stems from extracting LCUs from  
325 PLDNNs, which is negligible (less than 0.05s). Consequently, LinCon can scale up to large-sized  
326 neural networks.

327 **Tightness.** We compare the tightness of the over-approximation sets computed by different approaches. Figure 6 plots the experimental results, along with the corresponding simulation trajectories. For B2 and Tora, LinCon has a significant tightness improvement over Polar: the range of the over-approximation sets (red boxes) calculated by Polar far outreaches the range of the reachable states obtained from the simulation. Verisig 2.0 shares similar tightness results with Polar; see Appendix A.3 . We also defer to Appendix A.3 the experimental results for the remaining four cases where all three tools obtain similar tightness results.

328 **Discussion on CartPole and QUAD.** We discuss the verification results of the two complex cases,  
329 namely CartPole and QUAD. Both Polar and Verisig 2.0 fail to verify them. As shown in Figure 6,  
330 Polar aborts after 20 steps due to the huge over-approximation error in CartPole. In contrast, such  
331 a complex policy trained with our approach can be efficiently and tightly verified. In particular, the  
332 trajectories diverge first and finally merge. The computed reachable states tightly over-approximate  
333 these trajectories, and the computation takes only 151 seconds. Regarding the 12-dimensional QUAD  
334 case, Polar times out (two hours) after only two steps, while LinCon produces a very tight set of  
335 reachable states in 1054 seconds even after 15 steps, which is 7 times deeper than Polar. To the  
336 best of our knowledge, this is the first time that QUAD can be formally verified more than ten steps  
337 under various decision networks. Note that the trained policies used in the comparison differ due  
338 to different decision networks, and thus agents may follow different paths to the goal region (see the

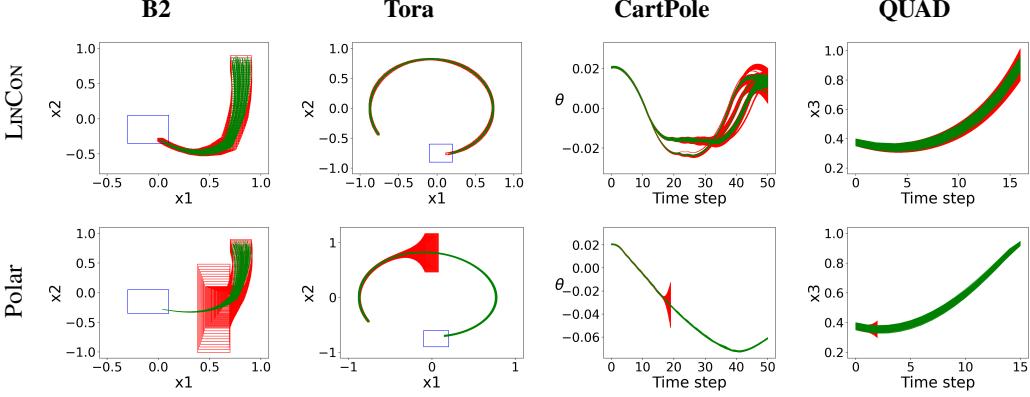


Figure 6: Tightness comparison with respect to the DRL systems with larger decision networks (red box: over-approximation sets; green lines: simulation trajectories; blue box: goal region).

345 simulated trajectories in Figure 6). Hence, for a fair comparison, we conduct more evaluations, from  
 346 which we can draw the same conclusion as from Figure 6. The results are given in Appendix A.3.

## 347 6 Related Work

348 **Policy Synthesis.** Several works adopt programmatic policies (e.g., decision trees and program  
 349 controllers) which are more interpretable and amenable to formal verification than neural policies.  
 350 Bastani et al. [15] construct a decision tree to represent a DNN policy based on imitation learning [41].  
 351 Verma et al. [42, 43] follow a similar routine, distilling neural network policies into predefined  
 352 program templates. Trivedi et al. [44] use a two-stage learning scheme to synthesize programmatic  
 353 policies. Some efforts are dedicated to exploring the combination of training and verification. Zhu et  
 354 al. [29] propose an inductive framework for synthesizing a deterministic policy program from neural  
 355 policies. Wang et al. [30] learn programmatic controllers based on verification feedback to avoid  
 356 safety violations. Our proposed PLDNN is essentially a DNN-based implementation of programmatic  
 357 controllers, which could be integrated with these verification-guided synthesis approaches.

358 **Reachability Analysis.** Our work is also built atop the approaches for reachability analysis of neural-  
 359 network-controlled systems. NNV [45] utilizes star set [46] to perform range analysis of decision  
 360 networks. JuliaReach [47] uses zonotope propagation to cover the output of a decision network.  
 361 Verisig [48, 9] models a decision network with differentiable activation functions (e.g., Tanh) as a  
 362 hybrid system and analyzes its reachability for over-approximating the network. ReachNN\* [27, 8]  
 363 abstracts the input-output mapping of a decision network with a Bernstein polynomial, together with  
 364 an error bound on the approximation. Sherlock [7] focuses on ReLU-based networks and computes  
 365 tight Taylor models via rule generation. Polar [10] integrates the Taylor and Bernstein approximation  
 366 techniques for building a Taylor model which over-approximates decision networks. All these efforts  
 367 over-approximate the embedded DNNs, which limit their scalability and verification accuracy.

## 368 7 Conclusion and Future Work

369 We have presented PLDNN that seamlessly integrates DNN and programmatic controls via state  
 370 abstraction for boosting the formal verification of DRL systems. Unlike traditional train-and-  
 371 transform approaches, PLDNN accompanies a novel inverse training and verification method, in  
 372 which a DNN is first transformed into an equivalent set of linear control policies and then trained  
 373 to optimize them. Experimental results have shown that PLDNN-controlled systems can be more  
 374 efficiently and tightly verified than DNN-based systems, with up to 438 times speedup and 7 times  
 375 deeper computation steps for a 12-dimensional control task.

376 Our work sheds light on a promising direction towards developing dependable DRL systems: learning  
 377 easy-to-verify and high-performance control policies via DRL and abstraction techniques. Given the  
 378 encouraging results of linear control policies, our work would also stimulate a passion for substituting  
 379 them with, e.g., polynomial control policies, for training and verifying more complex DRL systems.

380 **References**

- 381 [1] Z. Zhang and K. P. Lam, "Practical implementation and evaluation of deep reinforcement learning control  
382 for a radiant heating system," in *Proceedings of the 5th Conference on Systems for Built Environments*,  
383 2018, pp. 148–157.
- 384 [2] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving  
385 using end-to-end deep imitation learning," in *Robotics: science and systems*, 2018.
- 386 [3] X. Qi, Y. Luo, G. Wu, K. Boriboonsomsin, and M. Barth, "Deep reinforcement learning enabled self-  
387 learning control for energy efficient driving," *Transportation Research Part C: Emerging Technologies*,  
388 vol. 99, pp. 67–81, 2019.
- 389 [4] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep  
390 reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation  
391 Systems*, vol. 23, no. 6, pp. 4909–4926, 2021.
- 392 [5] F.-L. Fan, J. Xiong, M. Li, and G. Wang, "On interpretability of artificial neural networks: A survey," *IEEE  
393 Transactions on Radiation and Plasma Medical Sciences*, vol. 5, no. 6, pp. 741–760, 2021.
- 394 [6] R. V. Yampolskiy, "Unexplainability and incomprehensibility of AI," *Journal of Artificial Intelligence and  
395 Consciousness*, vol. 7, no. 02, pp. 277–291, 2020.
- 396 [7] S. Dutta, X. Chen, and S. Sankaranarayanan, "Reachability analysis for neural feedback systems using  
397 regressive polynomial rule inference," in *ACM International Conference on Hybrid Systems: Computation  
398 and Control (HSCC'19)*, 2019, pp. 157–168.
- 399 [8] J. Fan, C. Huang, X. Chen, W. Li, and Q. Zhu, "Reachnn\*: A tool for reachability analysis of neural-  
400 network controlled systems," in *International Symposium on Automated Technology for Verification and  
401 Analysis*. Springer, 2020, pp. 537–542.
- 402 [9] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. Pappas, and I. Lee, "Verisig 2.0: Verification of neural  
403 network controllers using taylor model preconditioning," in *International Conference on Computer Aided  
404 Verification (CAV'21)*. Springer, 2021, pp. 249–262.
- 405 [10] C. Huang, J. Fan, X. Chen, W. Li, and Q. Zhu, "Polar: A polynomial arithmetic framework for verifying  
406 neural-network controlled systems," in *The 20th International Symposium on Automated Technology for  
407 Verification and Analysis (ATVA'22)*. Springer, 2022, pp. 414–430.
- 408 [11] K. Makino and M. Berz, "Taylor models and other validated functional inclusion methods," *International  
409 Journal of Pure and Applied Mathematics*, vol. 6, pp. 239–316, 2003.
- 410 [12] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable  
411 guarantees," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp.  
412 2651–2659.
- 413 [13] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural  
414 networks," in *10th International Symposium on NASA Formal Methods (NFM'18)*. Springer, 2018, pp.  
415 121–138.
- 416 [14] L. M. Schmidt, G. Kontes, A. Plinge, and C. Mutschler, "Can you trust your autonomous car? interpretable  
417 and verifiably safe reinforcement learning," in *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE,  
418 2021, pp. 171–178.
- 419 [15] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," *Advances  
420 in Neural Information Processing Systems (NeurIPS'18)*, vol. 31, 2018.
- 421 [16] C. Buciluă, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM  
422 SIGKDD international conference on Knowledge discovery and data mining (KDD'06)*, 2006, pp. 535–541.
- 423 [17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint  
424 arXiv:1503.02531*, 2015.
- 425 [18] T. Brázdil, K. Chatterjee, M. Chmelík, V. Forejt, J. Křetínský, M. Kwiatkowska, D. Parker, and M. Ujma,  
426 "Verification of markov decision processes using learning algorithms," in *12th International Symposium on  
427 Automated Technology for Verification and Analysis (ATVA'14)*. Springer, 2014, pp. 98–114.
- 428 [19] D. Abel, D. E. Hershkowitz, and M. L. Littman, "Near optimal behavior via approximate state abstraction,"  
429 in *International Conference on Machine Learning (ICML'16)*, vol. 48, 2016, pp. 2915–2923.

- 430 [20] D. Abel, “A theory of state abstraction for reinforcement learning,” in *AAAI Conference on Artificial*  
 431 *Intelligence (AAAI’19)*, vol. 33, no. 01, 2019, pp. 9876–9877.
- 432 [21] P. Jin, J. Tian, D. Zhi, X. Wen, and M. Zhang, “Trainify: A CEGAR-driven training and verification frame-  
 433 work for safe deep reinforcement learning,” in *International Conference on Computer Aided Verification*  
 434 (*CAV’22*). Springer, 2022, pp. 193–218.
- 435 [22] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and  
 436 S. Yovine, “The algorithmic analysis of hybrid systems,” *Theoretical computer science*, vol. 138, no. 1, pp.  
 437 3–34, 1995.
- 438 [23] X. Chen, E. Ábrahám, and S. Sankaranarayanan, “Flow\*: An analyzer for non-linear hybrid systems,” in  
 439 *25th International Conference on Computer Aided Verification (CAV’13)*. Springer, 2013, pp. 258–263.
- 440 [24] P. Collins, D. Bresolin, L. Geretti, and T. Villa, “Computing the evolution of hybrid systems using rigorous  
 441 function calculus,” *IFAC Proceedings Volumes*, vol. 45, no. 9, pp. 284–290, 2012.
- 442 [25] M. Althoff, “An introduction to cora 2015.” *ARCH@ CPSWeek*, vol. 34, pp. 120–151, 2015.
- 443 [26] X. Chen, “Reachability analysis of non-linear hybrid systems using taylor models,” Ph.D. dissertation,  
 444 Fachgruppe Informatik, RWTH Aachen University, 2015.
- 445 [27] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu, “Reachnn: Reachability analysis of neural-network  
 446 controlled systems,” *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 5s, pp. 1–22, 2019.
- 447 [28] M. Grześ and D. Kudenko, “Multigrid reinforcement learning with reward shaping,” in *18th International*  
 448 *Conference Artificial Neural Networks (ICANN’08)*. Springer, 2008, pp. 357–366.
- 449 [29] H. Zhu, Z. Xiong, S. Magill, and S. Jagannathan, “An inductive synthesis framework for verifiable  
 450 reinforcement learning,” in *Proceedings of the 40th ACM SIGPLAN conference on Programming Language*  
 451 *Design and Implementation (PLDI’19)*, 2019, pp. 686–701.
- 452 [30] Y. Wang and H. Zhu, “Verification-guided programmatic controller synthesis,” in *29th International*  
 453 *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’23)*. Springer,  
 454 2023, pp. 229–250.
- 455 [31] P. Cousot and R. Cousot, “Abstract interpretation: a unified lattice model for static analysis of programs by  
 456 construction or approximation of fixpoints,” in *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium*  
 457 *on Principles of programming languages (POPL’77)*, 1977, pp. 238–252.
- 458 [32] M. Afzal, A. Asia, A. Chauhan, B. Chimdyalwar, P. Darke, A. Datar, S. Kumar, and R. Venkatesh, “Veriabs:  
 459 Verification by abstraction and test generation,” in *The 34th IEEE/ACM International Conference on*  
 460 *Automated Software Engineering (ASE’19)*. IEEE, 2019, pp. 1138–1141.
- 461 [33] K. Heo, H. Oh, and H. Yang, “Resource-aware program analysis via online abstraction coarsening,” in  
 462 *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE’19)*. IEEE, 2019, pp.  
 463 94–104.
- 464 [34] Z. Wang, A. Albargouthi, G. Prakriya, and S. Jha, “Interval universal approximation for neural networks,”  
 465 *Proceedings of the ACM on Programming Languages (POPL)*, vol. 6, no. POPL, pp. 1–29, 2022.
- 466 [35] E. Gallesty and P. Hokayem, “Lecture notes in nonlinear systems and control,” 2019.
- 467 [36] T. P. Lillicrap, J. J. Hunt, A. Pritzel *et al.*, “Continuous control with deep reinforcement learning,” in  
 468 *International Conference on Learning Representations (ICLR’16)*. OpenReview.net, 2016.
- 469 [37] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in  
 470 *International Conference on Machine Learning (ICML’18)*. PMLR, 2018, pp. 1587–1596.
- 471 [38] D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin, “Backpropagation: The basic theory,” *Backpropa-*  
 472 *gation: Theory, architectures and applications*, pp. 1–34, 1995.
- 473 [39] S.-i. Amari, “Backpropagation and stochastic gradient descent method,” *Neurocomputing*, vol. 5, no. 4–5,  
 474 pp. 185–196, 1993.
- 475 [40] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI  
 476 Gym,” 2016, arXiv:1606.01540.
- 477 [41] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *Proceedings of*  
 478 *the twenty-first International Conference on Machine Learning (ICML’04)*, 2004, p. 1.

- 479 [42] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, “Programmatically interpretable reinforcement  
480 learning,” in *International Conference on Machine Learning*. PMLR, 2018, pp. 5045–5054.
- 481 [43] A. Verma, H. Le, Y. Yue, and S. Chaudhuri, “Imitation-projected programmatic reinforcement learning,”  
482 *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- 483 [44] D. Trivedi, J. Zhang, S.-H. Sun, and J. J. Lim, “Learning to synthesize programs as interpretable and  
484 generalizable policies,” *Advances in Neural Information Processing Systems (NeurIPS’21)*, vol. 34, pp.  
485 25 146–25 163, 2021.
- 486 [45] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen *et al.*, “NNV: the neural network  
487 verification tool for deep neural networks and learning-enabled cyber-physical systems,” in *International  
488 Conference on Computer Aided Verification (CAV’20)*. Springer, 2020, pp. 3–17.
- 489 [46] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson, “Verification of deep convolutional neural networks using  
490 imagestars,” in *International Conference on Computer Aided Verification (CAV’20)*. Springer, 2020, pp.  
491 18–42.
- 492 [47] C. Schilling, M. Forets, and S. Guadalupe, “Verification of neural-network control systems by integrating  
493 taylor models and zonotopes,” in *AAAI Conference on Artificial Intelligence (AAAI’22)*, 2022, pp. 8169–  
494 8177.
- 495 [48] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, “Verifying the safety of autonomous  
496 systems with neural network controllers,” *ACM Transactions on Embedded Computing Systems*, vol. 20,  
497 no. 1, pp. 1–26, 2020.

498 **A Appendix**

499 **A.1 Benchmarks Setting**

500 We present the goal region and unsafe region of each benchmark in Table 3. For B1-B5 and Tora the  
 501 environment dynamics are the same with [9]. The dynamics of QUAD are the same with [10]. In  
 502 CartPole, we use the following ODEs to describe its dynamics in [40]:

$$\begin{aligned}\dot{x}_1 &= x_2, \quad \dot{x}_3 = x_4 \\ \dot{x}_2 &= (a + 0.05 * x_4 * x_4 * \sin(x_3)) / 1.1 - (0.05 * ((9.8 * \sin(x_3) - \cos(x_3)) * ((a + 0.05 * x_4 * x_4 * \sin(x_3)) / 1.1)) / \\ &\quad (0.5 * (4.0 / 3.0 - (0.1 * \cos(x_3) * \cos(x_3) / 1.1))) * \cos(x_3)) / 1.1 \\ \dot{x}_4 &= 9.8 * \sin(x_3) - \cos(x_3) * ((a + 0.05 * x_4 * x_4 * \sin(x_3)) / 1.1) / 0.5 * (4.0 / 3.0 - (0.1 * \cos(x_3) * \cos(x_3) / 1.1))\end{aligned}$$

503 For the 6 regular benchmarks, the target is training a policy to guide the agent to reach the goal region. Therefore,  
 504 we set a negative reward when the agent is not in the goal region. Once the agent reaches the goal region, it will  
 505 be awarded a positive reward. In CartPole, the target of training is to prevent the pole from falling over, namely  
 506 the pole angle  $-0.2 \leq x_3 \leq 0.2$ . We modify the original discrete reward function to a continuous reward function  
 507  $-|x_3|$  to try to balance the pole to stay upright. As for QUAD, we aim to control the altitude of the quadrotor  
 508 above 0.6 ( $x_3 \geq 0.6$ ). Thus we set a continuous reward function  $-|x_3 - 0.8|$  for training a policy that drives the  
 509 quadrotor to ascend above 0.6.

Table 3: Benchmarks Setting

Task	Initial Region	Goal Region	Unsafe Region
B1	$x_1 \in [0.8, 0.9]$ $x_2 \in [0.5, 0.6]$	$x_1 \in [0, 0.2]$ $x_2 \in [0.05, 0.3]$	—
B2	$x_1 \in [0.7, 0.9]$ $x_2 \in [0.7, 0.9]$	$x_1 \in [-0.3, 0.1]$ $x_2 \in [-0.35, 0.5]$	—
B3	$x_1 \in [0.8, 0.9]$ $x_2 \in [0.4, 0.5]$	$x_1 \in [0.2, 0.3]$ $x_2 \in [-0.3, -0.05]$	—
B4	$x_1 \in [0.25, 0.27]$ $x_2 \in [0.08, 0.1]$ $x_3 \in [0.25, 0.27]$	$x_1 \in [-0.05, 0.05]$ $x_2 \in [-0.05, 0]$	—
B5	$x_1 \in [0.38, 0.4]$ $x_2 \in [0.45, 0.47]$ $x_3 \in [0.25, 0.27]$	$x_1 \in [-0.4, -0.28]$ $x_2 \in [0.05, 0.22]$	—
Tora	$x_1 \in [-0.1, 0.2]$ $x_2 \in [-0.9, -0.6]$	$x_1 \in [-0.25, 0.10]$ $x_2 \in [0.2, 0.7]$	—
CartPole	$x_1, x_2, x_4 \in [0.02, 0.02]$ $x_3 \in [0.02, 0.021]$	—	$x_2 \in [26, 29]$ $x_3 \in [-0.2, 0.2]$
QUAD	$x_1, \dots, x_6 \in [0.35, 0.4]$ $x_7, \dots, x_{12} \in [0, 0]$	$x_3 > 0.6$	—

510 **A.2 Comparison on cumulative reward and robustness**

511 In this section, we provide the comparison results on cumulative reward and robustness of B3-B5 and Tora in  
 512 Figure 7. The solid lines and shadows refer to the average reward and 95% confidence interval, respectively.  
 513 For these four cases, it is obvious that there are comparable trends in the cumulative rewards of PLDNNs and  
 514 DNNs during training and under perturbation. Therefore, we can conclude that using PLDNNs will not affect  
 515 the training efficiency and the robustness of trained systems.

516 **A.3 Comparison on the tightness of over-approximation sets**

517 **Tightness results of regular cases.** We present the tightness comparison results of B1 and B3-B5 in Figure 8. For  
 518 these four regular cases, both Polar and Verisig 2.0 can produce tight over-approximation for decision networks,  
 519 thus all three methods achieve similar tightness results and successfully verify the goal-reach properties.

520 **Multiple evaluations on CartPole and QUAD.** Since the trained policies used in the comparison differ due  
 521 to different decision networks, we conduct more evaluations on these two complex cases. The corresponding  
 522 results are shown in Figure 10. For CartPole, Polar did not finish the calculation under three different networks  
 523 as depicted in Figure 10(d-f). Additionally, on the basis of the result of dealing with divergent traces as shown in

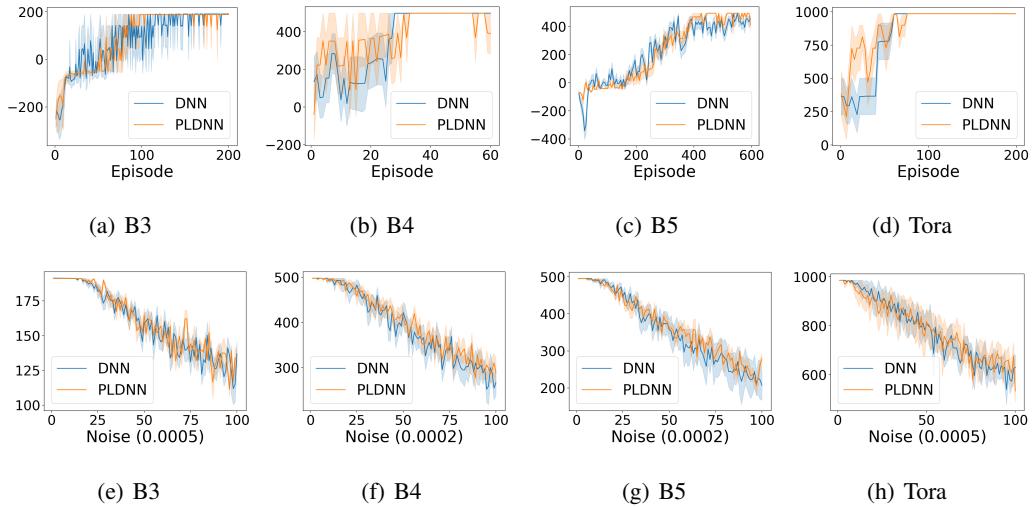


Figure 7: Performance (a-d) and robustness comparison (e-h) of the pldnn and DNN under the same settings. The number in the parentheses is the base of  $\sigma$ . For example, in B3 when the abscissa is equal to 50,  $\sigma = 50 \times 0.0005 = 0.025$ .

524 Figure 10(d), we can see that Polar is not suitable for dealing with the DNN-controlled systems with divergent  
 525 traces. As for QUAD, we record the computation results within 12 hours and obtain the results as shown in  
 526 Figure 10(j-l). We can observe that only a few time steps are completed by Polar, while LinCon can accomplish  
 527 more than 10 steps under multiple evaluations.

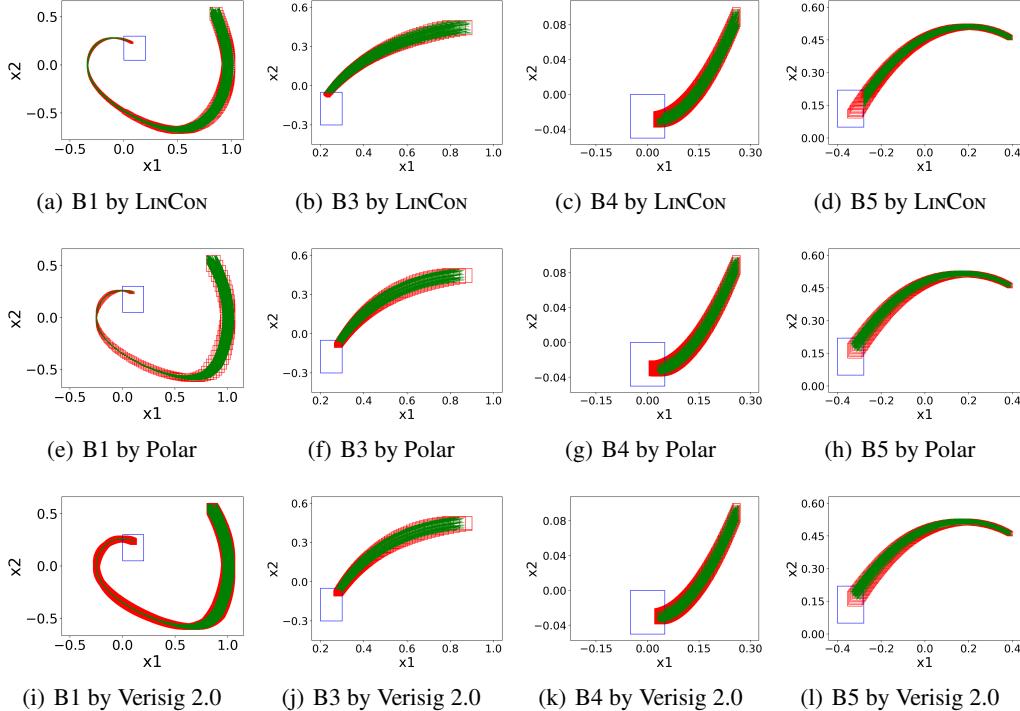


Figure 8: Tightness comparison on the DRL systems with larger decision networks (red box: over-approximation sets; green lines: simulation trajectories; blue box: goal region.)

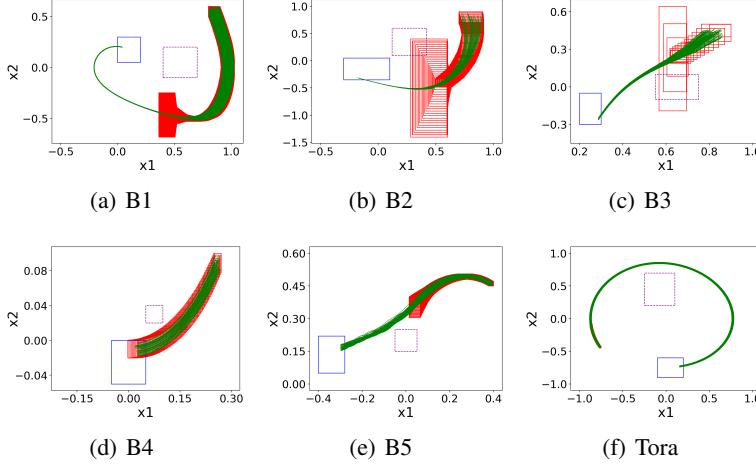


Figure 9: Assessing Verisig 2.0 on the larger networks with big weights. red box: over-approximation set; green lines: simulation trajectories; blue box: goal region; purple dashed box: unsafe region.

#### 528 A.4 Evaluation on Verisig 2.0 with Big Weights

529 Verisig 2.0 produces large over-approximation error when dealing with neural networks with big weights.  
 530 To demonstrate this, we initialize the weights of the neural network with larger values (random numbers  
 531  $w_l \sim N(\mu, \sigma^2)$  with  $\mu = 0, \sigma = 0.1$ ) and show the experimental results in Figure 9. We observe that the calculated  
 532 over-approximation sets contain large over-approximation error except for B4. In Tora, Verisig 2.0 fails to  
 533 calculate the complete reachable sets due to too large over-approximation error. Hence, it is fairly to say that  
 534 Verisig 2.0 is sensitive to the DNNs with big weights.

#### 535 A.5 Proofs for Theorem 1 and Theorem 2

536 **Definition 1** (Path of Hybrid Automaton [22]). *For a hybrid automaton  $H = \langle L, Var, Inv, F, T, G, R, I_0 \rangle$ , a*  
 537 *path of  $H$  is a finite or infinite state sequence starting from  $(l_0, v_0) \in Init$ :  $(l_0, v_0) \xrightarrow[\tilde{f}_{l_0}]{t_0} (l_1, v_1) \xrightarrow[\tilde{f}_{l_1}]{t_1} (l_2, v_2) \xrightarrow[\tilde{f}_{l_2}]{t_2} \dots$*   
 538  *$(l_3, v_3) \xrightarrow[\tilde{f}_{l_3}]{t_3} \dots$  such that:*

- 539 1.  $\forall 0 \leq t \leq t_i, \varphi_{\tilde{f}_{l_i}}(v_i, t) \in Inv(l_i)$
- 540 2.  $(l_i, l_{i+1}) \in Trans \wedge \varphi_{\tilde{f}_{l_i}}(v_i, t_i) \in Guard(l_i, l_{i+1}) \wedge v_{i+1} \in R(l_i, l_{i+1})$

541 **Definition 2** (Path of DRL systems). *Given a DRL system  $\mathcal{D}$  with an environment dynamics  $f$ , decision network  
 542  $\pi$ , time step size  $\delta$  and initial state set  $S_0$ , a path of  $\mathcal{D}$  is a finite or infinite state-action pair sequence:  $(s_0, a_0) \xrightarrow[\delta]{f} (s_1, a_1) \xrightarrow[\delta]{f} (s_2, a_2) \xrightarrow[\delta]{f} (s_3, a_3) \xrightarrow[\delta]{f} \dots$  such that:*

- 544 1.  $s_0 \in S_0$
- 545 2.  $[s_{i+1}, a_i] = \varphi_f([s_i, a_i], \delta) \wedge a_i = \pi(s_i)$

546 **Theorem 1** (Modeling DRL Systems as Hybrid Automata). *A DRL system with an environment dynamics  $f$ ,  
 547 decision network  $\pi$ , time step size  $\delta$  and initial state set  $S_0$ , can be modeled as the following hybrid automaton:*

- 548 •  $Var$ : state variable  $s$ , action  $a$ , clock variable  $t_c$       •  $I_0: \{(l_0, (s \in S_0, a = 0, t_c = \delta))\}$
- 549 •  $L: \{l_0\}$       •  $Inv: Inv(l_0) = \{s \in S, t_c \leq \delta\}$       •  $F: F(l_0) = \{\dot{s} = f(s, a), \dot{a} = 0, \dot{t}_c = 1\}$
- 550 •  $T: \{(l_0, l_0)\}$       •  $G: G(l_0, l_0) = \{t_c = \delta\}$       •  $R: R(l_0, l_0) = \{t_c = 0, a = \pi(s)\}$

551 *Proof.* We first analyze the path of the modeled hybrid automaton  $H$ . For an arbitrary initial state  
 552  $(l_0, [s_0, 0, \delta])$ ,  $s_0 \in S_0$ , since  $t_c = \delta$ , the only transition with guard condition  $\{t_c = \delta\}$  will be triggered and  
 553 transition to  $(l_0, [s_0, a_0, 0])$  where  $a_0 = \pi(s_0)$ . Then  $(l_0, [s_0, a_0, 0])$  will move to its time successor  $(l_0, [s_1, a_0, \delta])$   
 554 where  $[s_1, a_0] = \varphi_f([s_0, a_0], \delta)$ . Next, the transition with guard condition  $\{t_c = \delta\}$  will be made and conduct the

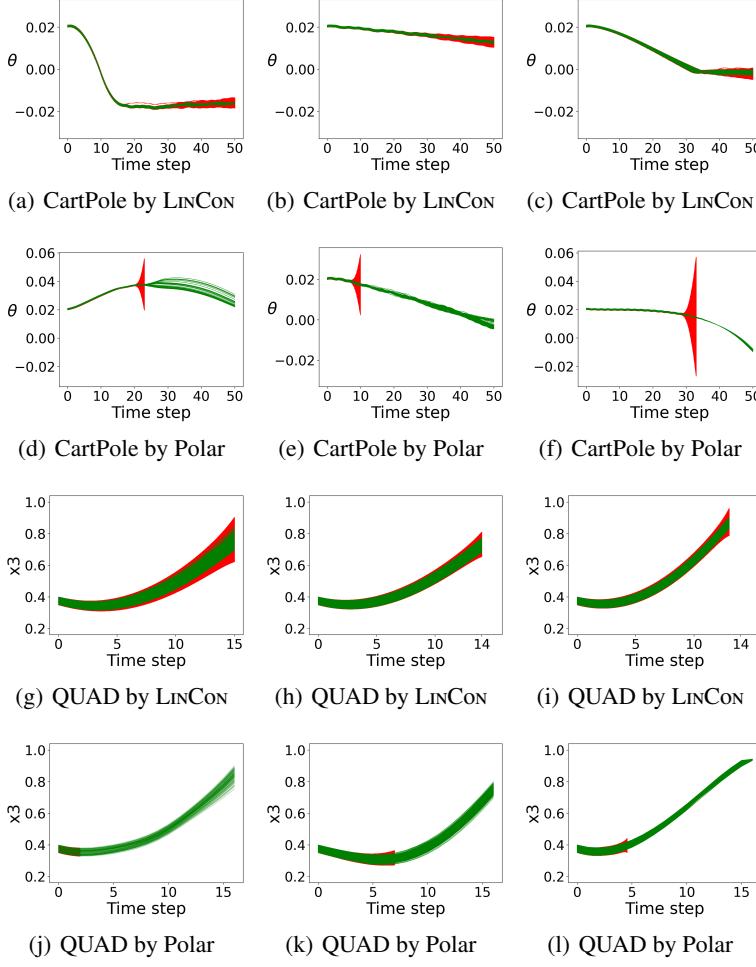


Figure 10: Multiple evaluations on CartPole and QUAD. red box: over-approximation set; green lines: simulation trajectories;

555 reset operation  $t_c = 0, a = \pi(s_1)$  to obtain the transition successor  $(l_0, [s_1, a_1, 0])$  in which  $a_1 = \pi(s_1)$ . Repeating  
556 the evolution to the time successor and the transition successor yields the following sequence:

$$(l_0, [s_0, 0, \delta]) \xrightarrow[f]{\delta} (l_0, [s_0, a_0, 0]) \xrightarrow[f]{\delta} (l_0, [s_1, a_1, 0]) \xrightarrow[f]{\delta} (l_0, [s_2, a_2, 0]) \xrightarrow[f]{\delta} \dots$$

557 where  $[s_{i+1}, a_i] = \varphi_f([s_i, a_i], \delta)$  and  $a_i = \pi(s_i)$ . According to Definition 2, given an arbitrary initial state  $s_0$ , at  
558 each time point  $i\delta$ , due to the same transition relation  $[s_{i+1}, a_i] = \varphi_f([s_i, a_i], \delta) \wedge a_i = \pi(s_i)$  in  $\mathcal{D}$  and the modeled  
559 hybrid automaton  $H$ , starting from same initial state  $s_0$ , the value of state variables and action of  $\mathcal{D}$  and  $H$  remain  
560 the same at each  $\delta$ . Then with the same state-action pair and dynamics  $f$  at each  $\delta$ ,  $\mathcal{D}$  and  $H$  also have the same  
561 state-action value during each  $\delta$ .  $\square$

562 **Theorem 2.** Given a DRL system with environment dynamics  $f$ , PLDNN  $\pi$ , time step size  $\delta$  and initial state set  
563  $S_0$ , it can be equivalently modeled as a hybrid automaton as follows:

- 564     • Var: state variable  $s$ , action  $a$ , clock variable  $t_c$      •  $I_0: \{(l_0, (s \in S_0, a = 0, t_c = \delta))\}$
- 565     •  $L: \{l_0\}$      • Inv:  $Inv(l_0) = \{s \in S, t_c \leq \delta\}$      •  $F: F(l_0) = \{\dot{s} = f(s, a), \dot{a} = 0, \dot{t}_c = 1\}$
- 566     •  $T: \{(l_0, l_0), \dots, (l_0, l_0)\}$  where  $|T| = |S_\phi|$
- 567     •  $G: G(T[i]) = \{t_c = \delta, s \in \phi^{-1}(s_\phi^i)\}$  where  $0 \leq i < |T| \wedge s_\phi^i \in S_\phi$
- 568     •  $R: R(T[i]) = \{t_c = 0, a = \pi_c(s) \cdot [1, s]\}$  where  $0 \leq i < |T| \wedge s \in \phi^{-1}(s_\phi^i)$

569 *Proof.* We prove that the path of hybrid automaton  $H_1$  in Theorem 1 is the same as the corresponding hybrid  
 570 automaton  $H_2$  in Theorem 2 for an arbitrary initial state  $(l_0, [s_0, a_0, \delta])$ . The path of  $H_2$  can be described by the  
 571 following sequence:

$$(l_0, [s_0, 0, \delta]) \xrightarrow[f]{0} (l_0, [s_0, a_0, 0]) \xrightarrow[f]{\delta} (l_0, [s_1, a_1, 0]) \xrightarrow[f]{\delta} (l_0, [s_2, a_2, 0]) \xrightarrow[f]{\delta} \dots$$

572 where  $[s_{i+1}, a_i] = \varphi_f([s_i, a_i], \delta)$  and  $a_i = \pi_c(s_i) \cdot [1, s_i]$ . With the dedicated structure of PLDNN  $\pi$ , we have  
 573  $\pi(s) = \pi_c(s) \cdot [1, s]$ . Thus, at each  $\delta$ , the discrete transitions of  $H_1$  and  $H_2$  change both location continuous  
 574 variables in the same way. We can conclude that the paths of  $H_1$  and  $H_2$  are exactly the same. Then based on  
 575 Theorem 1, it is obvious that  $\mathcal{D}$ ,  $H_1$  and  $H_2$  produce the same state-action value for the same initial state.  $\square$