Efficient LTL Model Checking of Deep Reinforcement Learning Systems using Policy Extraction

Peng Jin, Yang Wang, Min Zhang
Shanghai Key Laboratory for Trustworthy Computing, East China Normal University
Shanghai Trusted Industry Internet Software Collaborative Innovation Center
E-mail: `51194501007@stu.ecnu.edu.cn`, `{ywang,zhangmin}@sei.ecnu.edu.cn`

Abstract

Deep Reinforcement Learning (DRL) is a promising technology for solving intractable control tasks. Its applications in safety-critical fields require high-reliability guarantees. However, formal verification of DRL systems is challenging because deep neural networks (DNNs) embedded in the applications are uninterpretable. In this paper, we propose a novel approach to linear temporal logic (LTL) model checking of DRL systems by extracting interpretable policies from DNNs. The extracted policy can retain comparable performance to the original DNN. More importantly, its decision domain is finite and thus directly verifiable against LTL properties using existing model checking techniques. Experimental results on four classic control systems demonstrate the effectiveness of our approach.

1. Introduction

Deep Reinforcement Learning is being utilized heavily to solve diverse problems due to its strength in developing complex control systems [13, 19]. However, new risks emerge with this trend. The reliability of such systems is hard to guarantee because they cannot be formally verified like conventional systems. It becomes one of the significant obstacles to applying DRL in the real world [16, 17].

Three features make it a challenging problem to verifying DRL systems. First, the state space of such control systems is usually infinite and continuous, but most of the model checking-based approaches can only handle finite-state models [20]. Second, the system dynamics are generally nonlinear, which increases the complexity of formal verification [4]. Lastly, DNNs embedded in the systems are inexplicable, restricting the scalability of verification methods [9, 11]. Among them, the black-box nature of DNNs is the crux in defining faithful formal models for DRL systems, which are prerequisites for subsequent verification.

In this paper, we propose a simple but effective method to model checking LTL properties of DRL systems, which bypasses the crux by extracting interpretable policies from DNNs trained with popular DRL algorithms. Then systems driven by the extracted policies can be formally verified using existing model checking techniques. The policy can fulfill two essential requirements. One is that it can offer competitive performance compared to the original DNN. Besides, its decision domain is finite, where the decision-making unit (DMU) is a set of adjacent concrete states. For simplicity, we use DMU to represent the concrete states that it contains. We first discretize the state space of systems into finite DMUs and then determine the action adopted by each DMU to generate the final policy.

Based on extracted policies, we devise an algorithm to transform the DRL system into a finite-state transition system that can be model-checked via off-the-shelf tools such as *Spot* [7] against complex temporal properties. We apply our approach to verify four canonical control systems formally. Experimental results indicate that control systems driven by the extracted policies can be formally modeled and efficiently verified against complex temporal properties.

In summary, this paper makes three major contributions:

1. A novel method for extracting interpretable policies from trained DNNs.
2. An efficient model checking approach for verifying control systems driven by the extracted policies.
3. Four case studies for model checking the temporal properties of four control systems.

Paper organization. Section 2 introduces our policy extraction method. We present a model checking approach for control systems based on the extracted policies in Section 3. Section 4 shows the experimental results. Section 5 discusses related works, and Section 6 concludes the paper.

2. Interpretable Policy Extraction

In this section, we explain the notion of decision-making units (DMUs) used for policy extraction and present a state-space discretization method for generating DMUs and a process of determining corresponding actions for DMUs.

DOI reference number: 10.18293/SEKE2022-029

2.1. Decision-Making Unit

In DRL, policies encoded by DNNs map each concrete state to an optimal action. Therefore, the decision-making unit of such policies is infinite, making it difficult to build formal models for verification. Motivated by the works [2], we use finite axial bounding boxes to define the DMUs of our policy. All DMUs have the same area, and their union forms the state space. More importantly, they do not intersect, which indicates an assumption that concrete states within the DMU adopt the same action.

The assumption above to DMU is reasonable. Firstly, trained DNNs should take the same action for two inputs whose norm distance is close, which reflects the decision robustness [10, 22]. Besides, DRL handles the control systems that require continuous decision-making. Even if the chosen action is not optimal for all concrete states in the DMU, subsequent actions can compensate for the overall policy performance.

2.2. Generation of DMUs

We consider an n-dimensional state space S of the control system. Let L_i and U_i be the lower and upper bounds of the i-th dimension range of S, respectively. We introduce the discretization vector $D \in \mathbb{R}^n$ to divide S into DMUs, where $D = [d_1, \dots, d_n]$. For the i-th dimension range, it will be discretized into m_i $(= \lfloor \frac{U_i - L_i + d_i}{d_i} \rfloor)$ intervals, i.e. $[L_i, L_i + d_i), \dots, [L_i + m_i d_i - d_i, L_i + m_i d_i)$. Let $R_i = \{L_i, \dots, L_i + m_i d_i - d_i\}$ be the set of lower bounds of divided intervals in the i-th dimension. Then any DMU can be defined by a pair of two n-dimensional vectors $< [l_1, \dots, l_n], [d_1, \dots, d_n] >$, and its corresponding range is $[l_1, l_1 + d_1) \times \dots \times [l_n, l_n + d_n)$, where $l_i \in R_i$.

2.3. Action Determination

The action determination process is based on the DNNs trained by traditional DRL algorithms. The concrete state originally adopts the action output by the trained DNN. We aim to select the most frequently adopted action by concrete states in the DMU as its optimal action. However, there are infinite concrete states in the DMU, so it is infeasible to calculate precisely the action taken by most concrete states. Motivated by *randomized smoothing* [6] that uses Monte Carlo algorithms to evaluate the class that the trained DNN is most likely to classify for the images close to the input, we sample t concrete states in the DMU and determine its action based on the actions adopted by these concrete states. The sampled concrete state s is generated from n independent uniform distributions, where s_i is sampled from $U[l_i, l_i + d_i)$.

Let Act_a be the most frequently adopted action among t actions. If Act_a appears much more often than other actions, it will be the optimal action for the corresponding DMU. Otherwise, we repeat the sampling process until an explicit optimal action is obtained to avoid achieving a suboptimal action. When the action space of systems is continuous, we directly calculate the average of t actions as the final action of the DMU. In such a case, the action is represented by a vector of real numbers. The action vectors output by the trained DNN are hardly equal, so the actions adopted by sampled concrete states are always different.

Algorithm 1 Sampling-Based Action Determination

```
Input parameters: DMU s, action space $AS$.
Constant parameters: discretization vector $D$, sampling count $t$ and statistical significance $\alpha$.
  function DETERMINEACTION(s, $AS$)
      $States \leftarrow$ sample $t$ concrete states from s
      $Actions \leftarrow$ input $States$ to the trained DNN
      if TYPE($AS$) is continuous then
          return AVERAGE($Actions$)
      $Act_a, Act_b \leftarrow$ top two indices in $Actions$
      $Cnt_a, Cnt_b \leftarrow Actions[Act_a], Actions[Act_b]$
      if BINOMPVALUE($Cnt_a, Cnt_a + Cnt_b, 0.5) \leq \alpha$ then
          return $Act_a$
      else
          return DETERMINEACTION(s, $AS$)
```

Algorithm 1 depicts the whole of determining actions, where **s** denotes a DMU. We first sample t concrete states in the DMU and input them to the trained DNN to obtain actions that will be stored in the *Actions* array. For continuous action spaces, we directly take the average of actions. Otherwise, we choose the first two actions with the highest occurrences. Following the practice in [6], the BINOMPVALUE function returns the p-value of the hypothesis test, where $Cnt_a \sim \text{Binomial}(Cnt_a + Cnt_b, 0.5)$. If the p-value is less than or equal to the statistical significance α, Act_a is the action for the corresponding DMU. Namely, if Cnt_a is much higher than Cnt_b, Act_a will be returned. Otherwise, we will repeat the action determination process.

2.4. Hyperparameter Setting

Algorithm 1 requires three hyperparameters: sampling count t, statistical significance α, and discretization vector D. Usually, we set t to 5, which experimentally shows a good balance between sampling time cost and final policy performance. Besides, α is used to calibrate the resampling threshold. Its setting value is related to t. We set it to 0.2 so that only when Act_a appears 5 or 4 times can it be regarded as the action of the DMU. The vector D determines the granularity of generated DMUs. We can adjust it according to the performance of the extracted policy. For instance, if the extracted policy performs worse than the benchmark DNN, we can decrease it to generate preciser DMUs. Otherwise, we can increase it appropriately to reduce the number of DMUs, which can reduce the verification time to some extent.

3. Model Checking with Extracted Policies

The DMUs of extracted policies are finite and their union covers the entire state space. They can be treated as states of the transition system. Therefore, the system driven by the extracted policy can first be transformed into a DMU-based transition system (TS). Then existing model checking techniques can be leveraged to verify its LTL properties [3].

Accordingly, we introduce the abstract and refinement techniques to solve the problems encountered in constructing transition systems. Then we combine these two operations into an algorithm to construct TS automatically. At last, we discuss the process of LTL model checking based on the constructed TS.

3.1. Abstraction and Refinement

3.1.1 Abstraction in Building Transition Relations

Since the concrete states contained in the DMU adopt the same action, as demonstrated in Figure 1, we can treat them as a whole to carry out a state transition based on system dynamics. However, the generated area (pink part) may be irregular due to the nonlinearity of system dynamics, so it is hard to formally represent the reachable area, which brings difficulties to the construction of transition relations between DMUs.

Therefore, our goal is to abstract the generated region into the form we can represent. We properly expand the irregular area to fit multiple DMUs exactly. Formally, let min_i and max_i represent the minimum and maximum of the irregular area on the i-th dimension, respectively. We use $[l_1, u_1) \times \ldots \times [l_n, u_n)$ to define the corresponding expanded area. Then for each dimension, we can uniquely determine l_i and u_i based on the following constraints, where R_i is mentioned in Section 2.2:

$$l_i \le min_i,\ min_i < l_i + d_i,\ l_i \in R_i$$
$$max_i < u_i,\ u_i - d_i \le max_i,\ d_i | (u_i - l_i).$$

Let $count_i = (u_i - l_i)/d_i$. Obviously, the expanded area contains $\Pi_{i=1}^{n} count_i$ DMUs. In Figure 1, four successor DMUs intersect the irregular area. As for the subsequent transition, we can decompose it into the transition of each DMU that constitutes the expanded area, thus forming an iterative construction procedure.

3.1.2 Property-Based Refinement of DMUs

The LTL property consists of atomic propositions. Therefore, it is necessary to determine the atomic propositions each state satisfies in the transition system. Otherwise, LTL properties cannot be verified. However, treating DMUs as states in the transition system may lead to ambiguity since concrete states in the same DMU cannot be guaranteed to all satisfy certain atomic proposition. For example, for the atomic proposition b in Figure 2, partial concrete states in the DMU satisfy b, and the others satisfy $\neg b$, which will

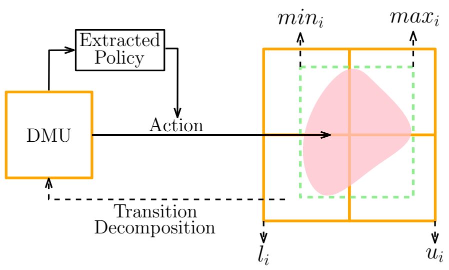

Figure 1: Computing direct successors of the DMU in a 2-dimensional control system.

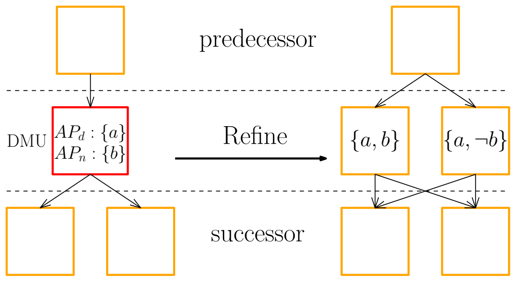

Figure 2: Property-based refinement of the DMU, where the red bounding box represents a nondeterministic DMU.

affect the verification of LTL properties like $F[\ b\ ]$. We call such DMUs and corresponding atomic propositions nondeterministic.

We refine the nondeterministic DMUs in the transition system constructed in Section 3.1.1 to eliminate ambiguity. In practice, we replace the nondeterministic DMU with multiple DMUs. Except for satisfying atomic propositions, the other properties of these substitute DMUs are the same as the original one, including the area, direct predecessors, and direct successors.

Formally, let AP_d define the set of propositions satisfied by the original DMU and $AP_n = \{\varphi_1, \ldots, \varphi_m\}$ represent the set of nondeterministic propositions. In Figure 2, AP_d and AP_n are $\{a\}$ and $\{b\}$, respectively. Then we define the set $AP_{sub} = \{\varphi_1, \neg\varphi_1\} \times \ldots \times \{\varphi_m, \neg\varphi_m\}$. Accordingly, there will be 2^m substitute DMUs. The i-th one satisfies the atomic propositions $AP_d \cup ap_i$, where $ap_i \in AP_{sub}$. If AP_n is empty, there will be only one substitute DMU that is exactly the original one.

3.2. Construction of Transition Systems

We combine the above two operations into an algorithm to construct the transition system, where the primary workflow is described in Algorithm 2.

We use $\mathbf{s}^0$ to denote the initial DMU and assume it is deterministic. For each DMU $\mathbf{s}$ fetched from *Queue*, we first determine its action based on Algorithm 1. Then we sequentially calculate the extreme values in each dimension of the irregular area generated by the state transition, i.e., min_i and max_i in Line 7. The set $\{\mathbf{s}^1, \ldots, \mathbf{s}^m\}$ represents the

Algorithm 2 Abstraction-Based TS Construction

Input: initial DMU $\mathbf{s}^0$, action space AS, state transition function f.
Output: transition system TS.

```
1:  $Queue \leftarrow \{\mathbf{s}^0\}$
2:  $TS$.setInitialState($\mathbf{s}^0$)
3:  while $Queue \neq \emptyset$ do
4:      Fetch $\mathbf{s}$ from $Queue$
5:      $action \leftarrow$ DETERMINEACTION($\mathbf{s}$, $AS$)
6:      for $i = 1, \ldots, n$ do
7:          $[min_i, max_i] \leftarrow$ CALEXTREMUM($f$, $\mathbf{s}$, $action$, $i$)
8:      $\{\mathbf{s}^1, \ldots, \mathbf{s}^m\} \leftarrow$ ABSTRACT($[min_1, max_1] \times \ldots \times [min_n, max_n]$)
9:      for $i = 1, \ldots, m$ do
10:         $\{\mathbf{s}^i_1, \ldots, \mathbf{s}^i_k\} \leftarrow$ REFINE($\mathbf{s}^i$)
11:         for $j = 1, \ldots, k$ do
12:             $TS$.addEdge($\mathbf{s} \rightarrow \mathbf{s}^i_j$)
13:             if $\mathbf{s}^i$ has not been traversed then
14:                 Push $\mathbf{s}^i_j$ into $Queue$
15: return $TS$
```

DMUs that constitute the expanded area. For each DMU $\mathbf{s}^i$ that is transitioned from $\mathbf{s}$, we replace it with multiple deterministic states and add the edge from $\mathbf{s}$ to the substitute DMUs into the transition system. Finally, if DMU $\mathbf{s}^i$ has not been traversed, we will push all its substitute DMUs into *Queue* to be visited, thus ensuring that they have the same direct successors.

3.3. LTL Model Checking

The verification algorithm for LTL properties first constructs the product of the automata that is equivalent to the negative form of the property and TS, then checks whether there exists a path that can be accepted [3]. Namely, TS is proved to satisfy the property if no violated path is found.

Since we preserve all original paths of the system and solve problems by adding the reachable range, we can ensure the soundness of verification results. Soundness means that the LTL properties verified on the constructed TS are also true in the original control system.

Therefore, all that remains is to leverage existing model checking tools after expressing the constructed TS in the specified rules. We utilize *Spot* [7] to complete the subsequent verification work in practice.

4 Experiments and Evaluation

We intend to demonstrate three aspects of our approach through experiments: (i) performance comparison with the benchmark DNN, (ii) LTL verification results of four systems after TS construction, and (iii) method effectiveness compared to relevant works.

All experiments are performed on a workstation running Ubuntu 18.04 with a 32-core AMD CPU. We select two control systems from Gym [5], plus Tora [12] and 4-Car Platoon [23], as test systems. We introduce them below:

Mountain Car (MC) A car is positioned on a one-dimensional track between two hills. It is expected to drive up the right mountain where the position is 0.5.

Pendulum (PD) A pole can rotate around a fixed endpoint, where it is expected to swing up and remain upright.

Tora A cart fixed on the wall with springs can move freely on a frictionless surface. The arm on the cart can rotate freely around an axis. The controller is expected to stabilize the system to an equilibrium state.

4-Car Platoon (4CP) Four cars are supposed to drive in a platoon behind each other. An intuitive requirement is that the four cars cannot collide.

4.1 Performance Comparison

We utilize a framework [21] to train benchmark neural networks. Note that the action space is discrete in MC and continuous in other systems, so we use DQN [15] to train DNNs in MC and DDPG [18] to train DNNs in the others. We use the default settings in the framework for all other training hyperparameters, such as learning rate. Besides, the discretization vector D set for policy extraction in each system is listed in Table 1 (Column: Initial DMU).

We compare the performance of extracted policies and original DNNs via the episode reward value. We test 500 episodes for both policies. As shown in Figure 3, we use boxplots to depict the distribution of test results, where the black dots are outliers.

We can see that the metrics describing the distribution of episodic rewards, such as minimum and median, are close. Significantly, the performance of extracted policies in MC and Tora is marginally better than the DNNs. Therefore, we can conclude that the policy extraction method is reasonable and practical for these control systems.

4.2. LTL Verification Analysis

The LTL verification results of four systems driven by extracted policies are listed in Table 1.

MC We set constraints for MC. Let $p(s)$ and $v(s)$ be the position and velocity of the car at state s, respectively. Property $G[\ p(s) = 0.2 \rightarrow v(s) > 0.01\ ]$ states that the car's speed must be greater than 0.01 at position 0.2. The other property is that the car's position will eventually be greater than 0.5, formulated by $F[\ p(s) > 0.5\ ]$.

Both properties can be verified under the initial state space $[-0.5, -0.4999) \times [0, 10^{-4})$. Since the TS is the same in both test cases, the verification times are also close.

PD We also set two LTL properties for PD. Firstly, we set $G[\ |\theta(s)| < \frac{\pi}{2}\ ]$, where $\theta(s)$ and $\omega(s)$ denote the angle and angular velocity of the pole, respectively. The formula describes that the pole's angle must always be in $(-\frac{\pi}{2}, \frac{\pi}{2})$. Besides, we expect that the angular velocity will eventually

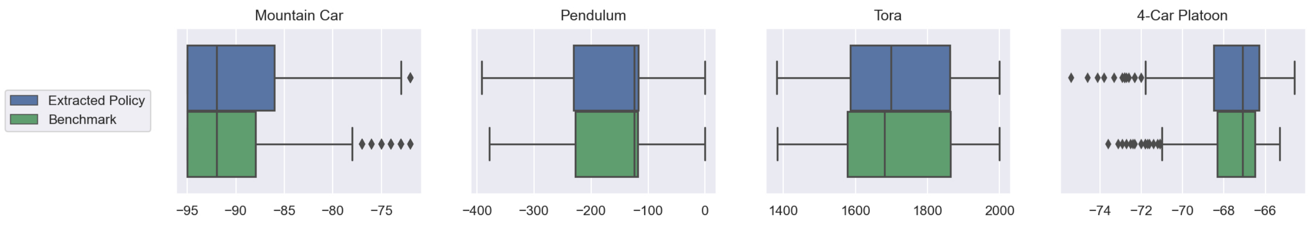

Figure 3: Performance comparison between the extracted policies and the DNNs (horizontal axis: the episode reward).

Table 1: LTL verification results of four systems driven by extracted policies.

Case	Initial DMU	LTL Property	Number	All	Verified	Time(s)
MC	$< [-0.5, 0.0], [10^{-4}, 10^{-4}] >$	$G[\ p(s) = 0.2 \to v(s) > 0.01\ ]$	1.2×10^6	✓	✓	2687
		$F[\ p(s) > 0.5\ ]$	1.2×10^6	✓	✓	2691
PD	$< [0, 0], [10^{-2}, 10^{-2}] >$	$G[\ \lvert\theta(s)\rvert < \frac{\pi}{2}\ ]$	728	✓	✓	5
		$G[\ \theta(s) < 0 \to F[\ \omega(s) > 0\ ]\ ]$	728	✓	✗	6
Tora	$< [0, 0, 0, 0], [10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}] >$	$G[\ \lvert s_1\rvert < 2 \wedge \lvert s_2\rvert < 2 \wedge \lvert s_3\rvert < 2 \wedge \lvert s_4\rvert < 2\ ]$	1.0×10^6	✗	✓	2731
4CP	$< [0, 0, 0, 0, 0, 0, 0], [10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}] >$	$G[\ d_1(s) > 0 \wedge d_2(s) > 0 \wedge d_3(s) > 0\ ]$	9721	✓	✓	154

Remark: the **Number** of DMUs in TS; whether TS contains **All** reachable DMUs; whether the property is **Verified**, and the verification **Time**.

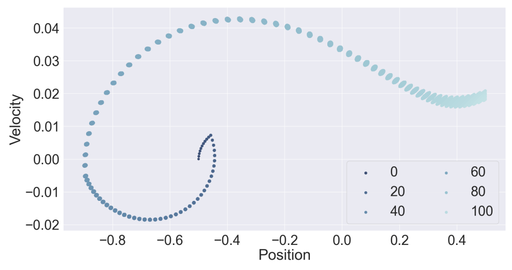

Figure 4: Traversal of DMUs over time steps in MC.

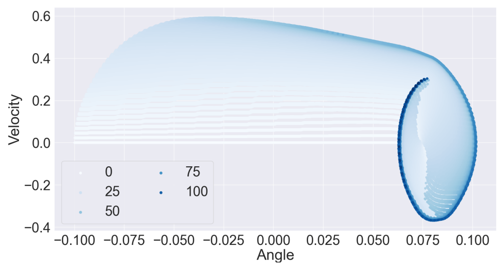

Figure 5: Traversal of DMUs over time steps in PD.

be greater than 0 if the angle is less than 0, i.e., $G[\ \theta(s) < 0 \to F[\ \omega(s) > 0\ ]\ ]$.

The second property fails to be verified because abstract methods applied in building transition relations add paths that violate the constraint.

Tora The equilibrium state for Tora is $[0, 0, 0, 0]$, so we expect the system can stay within $(-2, 2)^4$, which can be represented by $G[\ \lvert s_1\rvert < 2 \wedge \lvert s_2\rvert < 2 \wedge \lvert s_3\rvert < 2 \wedge \lvert s_4\rvert < 2\ ]$.

We limit the number of traversed DMUs to 10^6 to control the verification time. Therefore, the expected property can only be guaranteed within 24 time steps.

4CP We use $d_i(s)$ to denote the distance between the i-th car and $(i+1)$-th car, the constraint can be formulated as $G[\ d_1(s) > 0 \wedge d_2(s) > 0 \wedge d_3(s) > 0\ ]$.

Although 4CP is a 7-dimensional system, reachable DMUs are limited. The property can be verified among around 10^4 DMUs.

Evaluation Since the time required to build the transition system is much more than the execution time of *Spot*, the verification time is proportional to the number of traversed DMUs. However, compared with the state space, reachable DMUs are limited. For example, taking the center concrete state in the DMU, we depict the traversal of DMUs over time steps in MC and PD in Figure 4 and Figure 5, respectively, where different colors indicate that DMUs are traversed at different time steps. It can be seen that both systems operate in a fraction of the state space, which can make our method immune to the state explosion problem. Therefore, the combination of policy extraction and TS construction can ensure efficient verification of systems with limited reachable ranges.

Table 2: Performance comparison with ReachNN*.

Case	Neural Network	Ours	ReachNN*
B1	Tanh(2×20)	**389**	Unknown
	Tanh(2×100)	**401**	Unknown
B2	Sigmoid(2×20)	91	**22**
	Sigmoid(2×100)	**96**	105
MC	Sigmoid(2×16)	**2567**	Unknown
	Sigmoid(2×200)	**2574**	Unknown

4.3 Method Comparison

To our best knowledge, few related methods can verify properties other than *safety* and *liveness*, such as LTL in this paper. Therefore, we only compare the verification of *liveness* properties with ReachNN* [9].

For simplicity, we follow most of the experimental settings in [11], including test systems and verification properties. Therefore, we only list the necessary parameters in Table 2. There are two types of verification results: verification passes (verification time) or fails (Unknown). The discretization vector values in B1 and B2 are $[10^{-3}, 10^{-3}]$ and $[10^{-3}, 10^{-4}]$, respectively.

Our method outperforms ReachNN* in most cases. More importantly, the scalability of reachability analysis methods on which ReachNN* is based is limited by the structure and scale of DNNs. However, the trained DNN is a black box to our method via policy extraction, so the verification time is independent of DNNs.

5 Related Work

There is a growing literature on formal verification of DRL systems. Reachability analysis methods can calculate reachable sets of systems at each time step. For instance, Fan *et al.* approximated the DNN controller with Bernstein polynomials [9] while Ivanov *et al.* proposed a Taylor-model-based reachability algorithm to improve the scalability [11]. Besides, shielding can prevent DRL systems from exhibiting unsafe behavior [1, 23]. However, these methods can only verify the safety and liveness properties of DRL systems, while our approach can verify more complex temporal properties to guarantee applicability.

Works made by [8, 14] aim to model checking the DRL-driven systems via techniques on formal verification of DNNs. However, the scalability of their methods is limited by the size of neural networks, so the length of counterexamples obtained in experiments is limited. In contrast, the scalability of our method is DNN-independent.

6 Conclusion and Future Work

We proposed an LTL model checking approach to DRL verification. Our approach relies on extracting interpretable policies from trained DNNs and modeling DRL systems as a finite-state transition system using the extracted policies. It supports model checking of more complex temporal properties of DRL systems except for safety properties. We applied it to the DRL systems trained for four classic control problems. The experimental results show the effectiveness of our approach.

Our approach demonstrated the feasibility of extracting interpretable policies to substitute for inexplicable neural networks for formal verification. We plan to apply our approach to more complex DRL systems and devise more efficient model checking algorithms to improve scalability.

Acknowledgement

This work is supported by National Key Research Program (2020AAA0107800), Shanghai Science and Technology Commission (20DZ1100300), Shanghai Artificial Intelligence Innovation and Development Fund (2020-RGZN-02026), Shenzhen Institute of Artificial Intelligence and Robotics for Society (AC01202005020), NSFC projects (61872146, 62161146001). Yang Wang and Min Zhang are the corresponding authors.

References

[1] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, et al. Safe reinforcement learning via shielding. In *AAAI*, 2018.

[2] Edoardo Bacci and David Parker. Probabilistic guarantees for safe deep reinforcement learning. In *FORMATS*, pages 231–248, 2020.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[4] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *NIPS*, 31, 2018.

[5] Greg Brockman et al. OpenAI Gym, 2016.

[6] Jeremy Cohen et al. Certified adversarial robustness via randomized smoothing. In *ICML*, pages 1310–1320, 2019.

[7] Duret-Lutz et al. Spot 2.0—a framework for ltl and ω-automata manipulation. In *ATVA*. Springer, 2016.

[8] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 305–318, 2021.

[9] Jiameng Fan et al. Reachnn*: A tool for reachability analysis of neural-network controlled systems. In *ATVA*, pages 537–542, 2020.

[10] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[11] Radoslav Ivanov et al. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *International Conference on Computer Aided Verification*, 2021.

[12] Mrdjan Jankovic, Daniel Fontaine, and Petar V KokotoviC. Tora example: cascade-and passivity-based control designs. *IEEE Transactions on Control Systems Technology*, 4(3), 1996.

[13] Nathan Jay et al. Internet congestion control via deep reinforcement learning. *CoRR*, abs/1810.03259, 2018.

[14] Yafim Kazak, Clark Barrett, Guy Katz, and Michael Schapira. Verifying deep-rl-driven systems. In *Proceedings of the 2019 Workshop on Network Meets AI & ML*, pages 83–89, 2019.

[15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, et al. Continuous control with deep reinforcement learning. In *ICLR'16*, 2016.

[16] Yen-Chen Lin et al. Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint arXiv:1703.06748*, 2017.

[17] Björn Lütjens, Michael Everett, and Jonathan P How. Certified adversarial robustness for deep reinforcement learning. In *Conference on Robot Learning*, 2019.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[20] Pierre El Mqirmi, Francesco Belardinelli, and Borja G León. An abstraction-based method to check multi-agent deep reinforcement-learning behaviors. *arXiv preprint arXiv:2102.01434*, 2021.

[21] Kei Ota. TF2RL. https://github.com/keiohta/tf2rl/, 2020.

[22] Christian Szegedy et al. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[23] He Zhu et al. An inductive synthesis framework for verifiable reinforcement learning. In *PLDI*, pages 686–701, 2019.