Probabilistic Model Checking of Stochastic Reinforcement Learning Policies

Dennis Gross and Helge Spieker
Simula Research Laboratory, Norway

Keywords: Reinforcement Learning, Model Checking, Safety.

Abstract: We introduce a method to verify stochastic reinforcement learning (RL) policies. This approach is compatible with any RL algorithm as long as the algorithm and its corresponding environment collectively adhere to the Markov property. In this setting, the future state of the environment should depend solely on its current state and the action executed, independent of any previous states or actions. Our method integrates a verification technique, referred to as model checking, with RL, leveraging a Markov decision process, a trained RL policy, and a probabilistic computation tree logic (PCTL) formula to build a formal model that can be subsequently verified via the model checker Storm. We demonstrate our method's applicability across multiple benchmarks, comparing it to baseline methods called deterministic safety estimates and naive monolithic model checking. Our results show that our method is suited to verify stochastic RL policies.

1 INTRODUCTION

Reinforcement Learning (RL) has revolutionized the industry, enabling the creation of agents that can outperform humans in sequential decision-making tasks (Mnih et al., 2013a; Silver et al., 2016; Vinyals et al., 2019).

In general, an RL agent aims to learn a near-optimal policy to achieve a fixed objective by taking actions and receiving feedback through rewards and observations from the environment (Sutton and Barto, 2018). We call a policy *memoryless policy* if it only decides based on the current observation. A *deterministic policy* selects the same action in response to a specific observation every time, whereas a *stochastic policy* might select different actions when faced with the same observation. A popular choice is to employ a function approximator like a *neural network (NN)* to model such policies.

RL Problem. Unfortunately, learned policies are not guaranteed to avoid *unsafe behavior* (García and Fernández, 2015; Carr et al., 2023; Alshiekh et al., 2018). Generally, rewards lack the expressiveness to encode complex safety requirements (Littman et al., 2017; Hahn et al., 2019; Hasanbeig et al., 2020; Vamplew et al., 2022).

To resolve the issues mentioned above, verification methods like *model checking* (Baier and Katoen, 2008) are used to reason about the safety of RL, see for instance (Wang et al., 2020; Hasanbeig et al., 2020; Brázdil et al., 2014; Hahn et al., 2019). Model checking is not limited by properties that can be expressed by rewards but support a broader range of properties that can be expressed by *probabilistic computation tree logic* (PCTL; Hansson and Jonsson, 1994). At its core, model checking is a formal verification technique that uses mathematical models to verify the correctness of a system concerning a given (safety) property.

Hard to Verify. While existing research focuses on verifying stochastic RL policies (Bacci and Parker, 2020; Bacci et al., 2021; Bacci and Parker, 2022), these existing verification methods do not scale well with neural network policies that contain many layers and neurons.

Approach. This paper presents a method for verifying memoryless stochastic RL policies independent of the number of NN layers, neurons, or the specific memoryless RL algorithm.

Our approach hinges on three inputs: a Markov Decision Process (MDP) modeling the RL environment, a trained RL policy, and a probabilistic computation tree logic (PCTL) formula (Hansson and Jonsson, 1994) specifying the safety measurement. Using

Gross, D. and Spieker, H.
Probabilistic Model Checking of Stochastic Reinforcement Learning Policies.
DOI: 10.5220/0012357700003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 438-445
ISBN: 978-989-758-680-4; ISSN: 2184-433X

an incremental building process (Gross et al., 2022; Cassez et al., 2005; David et al., 2015) for the formal verification, we build only the reachable MDP portion by the trained policy. Utilizing Storm as our model checker (Hensel et al., 2022), we assess the policy's safety measurement, leveraging the constructed model and the PCTL formula.

Our method is evaluated across various RL benchmarks and compared to an alternative approach that only builds the part of the MDP that is reachable via the highest probability actions and an approach called naive monolithic model checking. The results confirm that our approach is suitable for verifying stochastic RL policies.

2 RELATED WORK

There exist a variety of related work focusing on the trained RL policy verification (Eliyahu et al., 2021; Kazak et al., 2019; Corsi et al., 2021; Dräger et al., 2015; Zhu et al., 2019; Jin et al., 2022). Bacci and Parker (2020) propose MOSAIC (MOdel SAfe Intelligent Control), which combines abstract interpretation and probabilistic verification to establish probabilistic guarantees of safe behavior. The authors model the system as a continuous-space discrete-time Markov process and focus on finite-horizon safety specifications. They tackle the challenges of infinite initial configurations and policy extraction from NN representation by constructing a finite-state abstraction as an MDP and using symbolic analysis of the NN. The *main difference* to our approach lies in constructing the induced discrete-time Markov chain (DTMC). Our approach verifies a specific probabilistic policy derived from an MDP. In contrast, the MOSAIC approach verifies safety guarantees by creating a finite-state abstraction of the MDP that accommodates the NN-based policy and focuses on finding safe regions of initial configurations. The abstraction must also extract the policy from its NN representation. Our approach only queries the trained policy and is not limited by the time horizon.

Bacci et al. (2021) presented the first technique for verifying if a NN policy controlling a dynamical system maintains the system within a safe region for an unbounded time. The authors use template polyhedra to overapproximate the reach set and formulate the problem of computing template polyhedra as an optimization problem. They introduce a MILP (Mixed-Integer Linear Programming) encoding for a sound abstraction of NNs with ReLU activation functions acting over discrete-time systems. Their method supports linear, piecewise linear, and non-linear systems with polynomial and transcendental functions. The safety verification verifies agents against a model of the environment. The main difference to our approach is that we are independent of the memoryless policy architecture and do not need to encode the verification problem as a MILP problem.

Bacci and Parker (2022) define a formal model of policy execution using continuous-state, finite-branching discrete-time Markov processes and build and solve sound abstractions of these models. The paper proposes a new abstraction based on interval Markov decision processes (IMDPs) to address the challenge of probabilistic policies specifying different action distributions across states. The authors present methods to construct IMDP abstractions using template polyhedra and MILP for reasoning about the NN policy encoding and the RL agent's environment. Furthermore, they introduce an iterative refinement approach based on sampling to improve the precision of the abstractions.

Gross et al. (2022) incrementally build a formal model of the trained RL policy and the environment. They then use the model checker Storm to verify the policy's behavior. They support memoryless stochastic policies by always choosing the action with the highest probability (deterministic safety estimation). In comparison, we build the model based on all possible actions with a probability greater than zero.

3 BACKGROUND

This section describes probabilistic model checking and investigates RL's details.

3.1 Probabilistic Model Checking

A *probability distribution* over a set X is a function $\mu\colon X \to [0,1]$ with $\sum_{x\in X}\mu(x) = 1$. The set of all distributions on X is denoted $Distr(X)$.

Definition 1 (Markov Decision Process)**.** *A* Markov decision process (MDP) *is a tuple* $M = (S, s_0, Act, Tr, rew, AP, L)$ *where* S *is a finite, nonempty set of states;* $s_0 \in S$ *is an initial state;* Act *is a finite set of actions;* $Tr\colon S \times Act \to Distr(S)$ *is a partial probability transition function;* $rew\colon S \times Act \to \mathbb{R}$ *is a reward function;* AP *is a set of atomic propositions;* $L\colon S \to 2^{AP}$ *is a labeling function that assigns atomic propositions to states.*

We employ a factored state representation where each state s is a vector of features $(f_1, f_2, ..., f_d)$ where each feature $f_i \in \mathbb{Z}$ for $1 \leq i \leq d$ (d is the dimension of the state). Furthermore, we denote $Tr(s,a)(s')$ with

$Tr(s,a,s')$ and $Tr(s,a,s')$ can be written as $s \xrightarrow{a} s'$. If $|\,Act(s)\,| = 1$, we can omit the action a in $Tr(s,a,s')$ and write $Tr(s,s')$. Also, we define $NEIGH(s)$ for the set of all states $s' \in S$ that $Tr(s,s') \neq 0$.

The available actions in $s \in S$ are $Act(s) = \{a \in Act \mid Tr(s,a) \neq \bot\}$ where $Tr(s,a) \neq \bot$ is defined as action a at state s does not have a transition (action a is not available in state s). An MDP with only one action per state ($\forall s \in S : |\,Act(s)\,| = 1$) is a DTMC D.

Definition 2 (Discrete-time Markov Chain)**.** *A discrete-time Markov chain (DTMC) is an MDP such that $|\,Act(s)\,| = 1$ for all states $s \in S$. We denote a DTMC as a tuple $M_D = (S, s_0, Tr, rew)$ with S, s_0, rew as in Definition 1 and transition probability function $Tr : S \to Distr(S)$.*

When an agent is executed in an environment modeled as an MDP, a *policy* is a rule that an agent follows in deciding which action to take based on its current observation of the state of the environment to optimize its objective.

Definition 3 (Deterministic Policy)**.** *A memoryless deterministic policy for an MDP $M=(S, s_0, Act, Tr, rew)$ is a function $\pi\colon S \to Act$ that maps a state $s \in S$ to action $a \in Act$.*

Applying a policy π to an MDP M yields an *induced DTMC D* where all non-determinisms within the system are resolved.

Definition 4 (Stochastic Policy)**.** *A memoryless stochastic policy for an MDP $M=(S, s_0, Act, T, rew)$ is a function $\hat{\pi}\colon S \to Distr(Act)$ that maps a state $s \in S$ to distribution over actions $Distr(Act)$.*

Permissive policies facilitate a richer exploration of the state space by allowing the selection of multiple actions at each observation, potentially uncovering nuanced policies.

Definition 5 (Permissive Policy)**.** *A permissive policy $\tau\colon S \to 2^{Act}$ selects multiple actions in every state.*

We specify the properties of a DTMC via the specification language PCTL (Wang et al., 2020).

Definition 6 (PCTL Syntax)**.** *Let AP be a set of atomic propositions. The following grammar defines a state formula: $\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid P_{\bowtie p} \mid P^{max}_{\bowtie p}(\phi) \mid P^{min}_{\bowtie p}(\phi)$ where $a \in AP, \bowtie \in \{<, >, \leq, \geq\}$, $p \in [0,1]$ is a threshold, and ϕ is a path formula which is formed according to the following grammar*

$$\phi ::= X\Phi \mid \phi_1\, U\, \phi_2 \mid \phi_1\, F_{\theta_i t}\, \phi_2 \;\; with\; \theta_i = \{<, \leq\}.$$

We further define "eventually" as $\Diamond\phi := true\ U\ \phi$.

For MDPs, PCTL formulae are interpreted over the states of the induced DTMC of an MDP and a policy. In a slight abuse of notation, we use PCTL state formulas to denote probability values. That is, we sometimes write $P_{\bowtie p}(\phi)$ where we omit the threshold p. For instance, in this paper, $P(\Diamond coll)$ denotes the reachability probability of eventually running into a collision. There exist a variety of model checking algorithms for verifying PCTL properties (Courcoubetis and Yannakakis, 1988, 1995). PRISM (Kwiatkowska et al., 2011) and Storm (Hensel et al., 2022) offer efficient and mature tool support for verifying probabilistic systems.

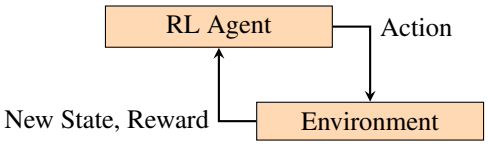

Figure 1: This diagram represents an RL system in which an agent interacts with an environment. The agent receives a state and a reward from the environment based on its previous action. The agent then uses this information to select the next action, which it sends to the environment.

3.2 Reinforcement Learning

For MDPs with many states and transitions, obtaining optimal policies π^* is difficult. The standard learning goal for RL is to learn a policy π in an MDP such that π maximizes the accumulated discounted reward, that is, $\mathbb{E}[\sum_{t=0}^{N} \gamma^t R_t]$, where γ with $0 \leq \gamma \leq 1$ is the discount factor, R_t is the reward at time t, and N is the total number of steps (see Figure 1). In RL, an agent learns through interaction with its environment to maximize a reward signal, via RL algorithm such as deep Q-learning (Mnih et al., 2013b). Note that rewards lack the expressiveness to encode complex safety requirements which can be specified via PCTL (Littman et al., 2017; Hahn et al., 2019; Hasanbeig et al., 2020; Vamplew et al., 2022). Therefore, model checking is needed to verify that the trained RL policies satisfy the complex requirements.

4 METHODOLOGY

In this section, we introduce a method for verifying the safety of stochastic RL policies via probabilistic model checking concerning a safety measurement. Given the MDP M of the environment, a trained RL policy $\hat{\pi}$, and a safety measurement m, the general workflow is as follows and will be detailed below:

1. **Induced MDP Construction**: We construct a new MDP using the original MDP and the trained RL policy $\hat{\pi}$. This new MDP includes only the states that are reachable by the policy and actions

that the policy considers, i.e., those for which the policy selection probability is greater than zero.

2. **Induced DTMC Transformation**: The newly constructed MDP $\hat{M}$ is transformed into an induced DTMC, denoted as $\hat{D}$. The transition probabilities between states in $\hat{D}$ are updated based on both the modified MDP $\hat{M}$ and the action distribution of the original RL policy $\hat{\pi}$.
3. **Safety Verification**: Finally, the safety measurement m of $\hat{D}$ is rigorously verified using the Storm model checker.

Running Example. To elucidate the methodology, let's consider a running example featuring an MDP that models a grid-like environment (see Figure 2). In this environment, an RL agent can navigate among four discrete states, denoted as $x = 1$, $x = 2$, $x = 3$, and $x = 4$, corresponding to positions A, B, C, and D, respectively. From each state, the agent can execute one of three actions: UP, NOP (No Operation), or DOWN. The transition probabilities between states are defined based on the chosen actions. For instance, executing the UP action from state A ($x = 1$) gives the agent a 0.2 probability of transitioning to state B ($x = 2$) and a 0.8 probability of transitioning to state C ($x = 3$). For simplicity, let's assume one reward for each state-action pair. This example functions as a foundational MDP against which we will test our verification method.

4.1 Induced MDP Construction

To perform the verification of stochastic RL policies, we initially construct an induced MDP $\hat{M}$ from a given MDP M (as shown in Figure 2), a trained stochastic RL policy $\hat{\pi}$, and a safety measurement m.

We commence the construction process at the initial state s_0 of the original MDP M. Starting from the initial state s_0, we iteratively visit each state s that is reachable under the policy $\hat{\pi}$ by only including actions a for which $\hat{\pi}(a|s) > 0$. This creates an MDP $\hat{M}$ induced by a permissive policy τ and ensures that $\hat{M}$ is reduced to those states and actions that are reachable by $\hat{\pi}$ (see Figure 3). This induced MDP serves as the basis for the subsequent transformation into an induced DTMC $\hat{D}$ for formal verification.

4.2 Induced DTMC Transformation

To convert the induced MDP $\hat{M}$ into an induced DTMC $\hat{D}$, we perform a probabilistic transformation on the state transitions (refer to Figure 4). Specifically, we update the transition function $\hat{Tr}_{\hat{D}}(s,s')$,

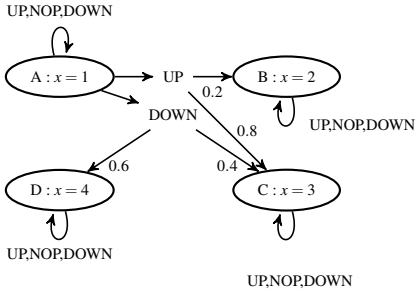

Figure 2: The MDP M with $S = \{x = 1, x = 2, x = 3, x = 4\}$, A={UP,NOP,DOWN}, and rew: $S \times A \to \{1\}$.

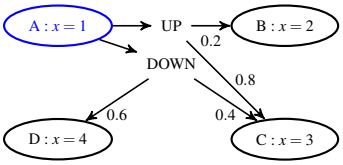

Figure 3: Induced MDP $\hat{M}$. We add the state-action transition to the MDP for each action with a probability greater than zero $\hat{\pi}(a|s)$. In this example, the chosen actions are UP and DOWN with the corresponding probabilities $\hat{\pi}(UP|x = 1) = 0.3$ and $\hat{\pi}(DOWN|x = 1) = 0.7$ at state $x = 1$.

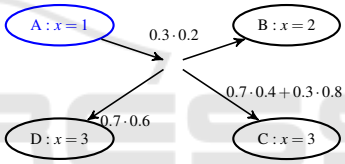

Figure 4: Transitions update for $\hat{D}$. For example, $\hat{Tr}_{\hat{D}}(x = 1)(x = 2) = Tr(x = 1, UP, x = 2) \cdot \hat{\pi}(UP \mid x = 1) = 0.2 \cdot 0.3 = 0.06$ at state $x = 1$. Repeat with step from Figure 3 for all reachable states by the trained RL policy.

which defines the probability of transitioning from state s to state s', for each s' that is a neighboring state $NEIGH(s)$ of s within the set of all states $S_{\hat{M}}$ of the induced MDP $\hat{M}$.

The transition function $\hat{Tr}_{\hat{D}}(s,s')$ is given by the following equation:

$$\hat{Tr}_{\hat{D}}(s,s') = \sum_{a \in Act_{\hat{M}}(s)} Tr_{\hat{M}}(s,a,s')\hat{\pi}(a|s)$$

Where $Act_{\hat{M}}(s)$ represents the set of actions available in state s of $\hat{M}$, $Tr_{\hat{M}}(s,a,s')$ denotes the original transition probability of moving from state s to state s' when action a is taken in $\hat{M}$, and $\hat{\pi}(a|s)$ is the action-selection probability of action a in state s under the RL policy $\hat{\pi}$. The equation essentially performs a weighted sum of all possible transitions from state s to state s', using the probabilities of selecting each action a in $\hat{M}$ according to $\hat{\pi}$ as the weights. This transformation inherently incorporates the decision-making behavior of the RL policy into the probabilistic structure of $\hat{D}$, enabling us to perform verification tasks.

Table 1: Trained PPO Policies. The learning rates are 0.0001, the batch sizes are 32, and the seeds are 128. "Ep." is shorthand for "episode." The reward is averaged over a sliding window of 100 episodes.

Environment	Layers	Neurons	Ep.	Reward
Freeway	2	64	898	0.78
Crazy Climber	2	1024	3,229	76.22
Avoidance	2	64	7,749	6,194

4.3 Safety Verification

Once the induced DTMC $\hat{D}$ is constructed, the final step in our methodology is to do the safety measurement using the Storm model checker. It takes as input the DTMC $\hat{D}$ and a safety measurement m. Upon feeding $\hat{D}$ and the specified safety measurement m into Storm, the model checker systematically explores all possible states and transitions in $\hat{D}$ to return the safety measurement result.

4.4 Limitations

Our method is independent of the RL algorithm; therefore, we can verify any neural network size. However, it requires the policy to obey the Markov property, i.e., there is no memory, and the action is selected only using the current state. Furthermore, we are limited to discrete state and action spaces due to the discrete nature of the model checking component.

5 EXPERIMENTS

We now evaluate our method in multiple RL environments. We focus on the following research questions:

- Can we model check RL policies in common benchmarks such as Freeway?
- How does our approach compare to COOL-MC's deterministic estimation and naive monolithic model checking?
- How does the method scale?

The experiments are performed by initially training the RL policies using the PPO algorithm (Barhate, 2021), then using the trained policies to answer our research questions.

5.1 Setup

We now describe our setup. First, we describe the used environments, then the trained policies, and finally, our technical setup.

5.1.1 Environments

We focus on environments that have previously been used in the RL literature.

Freeway. The RL agent controls a chicken (up, down, no operation) running across a highway filled with traffic to get to the other side. Every time the chicken gets across the highway, it earns a reward of one. An episode ends if the chicken gets hit by a car or reaches the other side. Each state is an image of the game's state. Note that we use an abstraction of the original game, which sets the chicken into the middle column of the screen and contains fewer pixels than the original game, but uses the same reward function and actions (Mnih et al., 2015).

Avoidance. This environment contains one agent and two moving obstacles in a two-dimensional grid world. The environment terminates when a collision between the agent and an obstacle happens. For each step that did not resolve in a collision, the agent gets rewarded with a reward of 100. The environment contains a slickness parameter, which defines the probability that the agent stays in the same cell (Gross et al., 2022).

Crazy Climber. It is a game where the player has to climb a wall (Mnih et al., 2015). This environment is a PRISM abstraction based on this game. Each state is an image. A pixel with a One indicates the player's position. A pixel with a Zero indicates an empty pixel. A pixel with a Three indicates a falling object. A pixel with a four indicates the player's collision with an object. The right side of the wall consists of a window front. The player must avoid climbing up there since the windows are unstable. For every level the play climbs, the player gets a reward of 1. To avoid falling obstacles, the player has the option to move left, move right, or stay idle.

5.1.2 Trained RL Policies

We trained PPO agents in the previously introduced RL environments. The RL training results are summarized in Table 1.

5.1.3 Technical Setup

We executed our benchmarks in a docker container with 16 GB RAM, and an AMD Ryzen 7 7735hs with Radeon graphics x 16 processor with the operating system Ubuntu 20.04.5 LTS. For model checking, we use Storm 1.7.1 (dev) (Hensel et al., 2022).

Table 2: RL benchmarks across various environments and metrics. Columns display the number of built states, transitions, safety measures, and computational time required. The time is calculated by adding the time taken to build the model to the time taken to check it and is expressed in seconds. Notably, the time for model checking is negligible and, therefore, not explicitly mentioned.

Setup		Deterministic Policy				Stochastic Policy				Naive Monolithic (No RL Policy)			
Environment	**Measurement**	**States**	**Transitions**	**Result**	**Time**	**States**	**Transitions**	**Result**	**Time**	**States**	**Transitions**	**Result**	**Time**
Freeway	$P(\lozenge \text{goal})$	123	340	0.7	2.5	496	2,000	0.7	11	496	2800	1.0	0.06
Freeway	$P(\text{mid U mid}_{-1})$	21	36	1.0	0.5	272	1,120	1.0	6	272	1552	1.0	0.07
Crazy Climber	$P(\lozenge \text{coll})$	8,192	32,768	0.0	146	40,960	270,336	1.0	1,313	40,960	385,024	1.0	0.4
Avoidance	$P(\lozenge_{\leq 100} \text{coll})$	625	8698	0.84	1.5	15,625	892,165	0.84	1,325	15,625	1,529,185	1.0	2.0

5.2 Analysis

We now go through our evaluation and answer our research questions.

5.2.1 Can We Verify Stochastic RL Policies in Common Benchmarks Such as Freeway?

In this experiment, we investigate the applicability of our approach to evaluating RL policies, specifically focusing on the Freeway environment.

Execution. Our analysis encompasses two dimensions: (1) the probability of reaching the desired goal, denoted by $P(\lozenge \text{goal})$, and (2) the complex behavior involving the likelihood of the agent reverting to its starting position, denoted by $P(\text{mid} \, U \, \text{mid}_{-1})$.

Results. For the first dimension, we evaluated the probability of the trained Freeway RL agent successfully crossing the street without colliding with a car. The model checking result for this safety measurement yielded $P(\lozenge \text{goal}) = 0.7$, indicating that the agent has a 70% chance of safely reaching the other side of the road.

For the second dimension, we examined the agent's likelihood of reversing direction and returning to the starting position after having entered the street. The model checking process, in this case, revealed that $P(\text{mid U mid}_{-1}) = 1$, confirming that the agent will indeed return to the starting position with certainty under the analyzed conditions.

In summary, our findings demonstrate that it is not only feasible to apply model checking methods to stochastic RL policies in commonly used benchmarks like Freeway but also that these methods are versatile enough to evaluate complex safety requirements.

5.2.2 How Does Our Approach Compare to COOL-MC's Deterministic Estimation and Naive Monolithic Model Checking?

This experiment compares our approach with *deterministic estimations of the safety measures of trained policies* and *naive monolithic model checking*. *Deterministic estimation* incrementally builds the induced DTMC by querying the policy at every state for the highest probability action (Gross et al., 2022). *Naive monolithic model checking* is called "naive" because it does not take into account the complexity of the system or the number of possible states it can be in, and it is called "monolithic" because it treats the entire system as a single entity, without considering the individual components of the system or the interactions between them (Gross et al., 2023a). In short, it does not verify the trained RL policy but rather the overall environment. For instance, $P(\lozenge goal) = 1$ indicates a path to reach the goal state eventually.

Execution. We use the trained stochastic policy to build and verify the induced DTMCs correspondingly for the deterministic estimation and our approach. For naive monolithic model checking, we build the whole MDP and verify it correspondingly.

Results. The data presented in Table 2 indicates that our method yields precise results (see Crazy Climber). In contrast, the deterministic estimation technique exhibits faster performance. This is due to its method of extending only the action transitions associated with the highest-probability action. The naive monolithic model checking results are bounds and do not reflect the actual RL policy performance. In the context of naive monolithic model checking, the number of states and transitions is larger than the other two approaches. This indicates that naive monolithic model checking runs faster out of memory than the other two approaches, which is critical in environments with many states and transitions (Gross et al., 2023b).

5.2.3 How Does the Method Scale?

Based on our experiments, detailed in Table 2, we find that the primary limitations of our approach stem from the increasing number of states and transitions. Given the probabilistic characteristics inherent in the RL policy, there is an increase in the number of states

and transitions that can be reached compared to a deterministic RL policy. This expansion results in more extended times for the overall model checking. Optimizing our method's incremental building process may increase the model checking performance (Gross et al., 2023a).

6 CONCLUSION

We presented a methodology for verifying memoryless stochastic RL policies, thus addressing a gap in the current body of research regarding the safety verification of RL policies with complex, layered NNs. Our method operates independently of the specific RL algorithm in use. Furthermore, we demonstrated the effectiveness of our approach across various RL benchmarks, confirming its capability to verify stochastic RL policies comprehensively.

For future work, exploring the integration of safe RL (Carr et al., 2023) and stochastic RL verification offers a promising path to validate policies' reliability and enhance their operational safety across diverse environments. Additionally, merging stochastic RL verification with interpretability (Zhao et al., 2023) and explainability (Vouros, 2023) approaches could significantly bolster the understanding of RL policies.

REFERENCES

Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., and Topcu, U. (2018). Safe reinforcement learning via shielding. In *AAAI*, pages 2669–2678. AAAI Press.

Bacci, E., Giacobbe, M., and Parker, D. (2021). Verifying reinforcement learning up to infinity. In *IJCAI*, pages 2154–2160. ijcai.org.

Bacci, E. and Parker, D. (2020). Probabilistic guarantees for safe deep reinforcement learning. In *FORMATS*, volume 12288 of *Lecture Notes in Computer Science*, pages 231–248. Springer.

Bacci, E. and Parker, D. (2022). Verified probabilistic policies for deep reinforcement learning. In *NFM*, volume 13260 of *Lecture Notes in Computer Science*, pages 193–212. Springer.

Baier, C. and Katoen, J. (2008). *Principles of model checking*. MIT Press.

Barhate, N. (2021). Minimal pytorch implementation of proximal policy optimization. https://github.com/nikhilbarhate99/PPO-PyTorch.

Brázdil, T., Chatterjee, K., Chmelik, M., Forejt, V., Kretínský, J., Kwiatkowska, M. Z., Parker, D., and Ujma, M. (2014). Verification of markov decision processes using learning algorithms. In *ATVA*, volume 8837 of *Lecture Notes in Computer Science*, pages 98–114. Springer.

Carr, S., Jansen, N., Junges, S., and Topcu, U. (2023). Safe reinforcement learning via shielding under partial observability. In *AAAI*, pages 14748–14756. AAAI Press.

Cassez, F., David, A., Fleury, E., Larsen, K. G., and Lime, D. (2005). Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer.

Corsi, D., Marchesini, E., and Farinelli, A. (2021). Formal verification of neural networks for safety-critical tasks in deep reinforcement learning. In de Campos, C. and Maathuis, M. H., editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 333–343. PMLR.

Courcoubetis, C. and Yannakakis, M. (1988). Verifying temporal properties of finite-state probabilistic programs. In *FOCS*, pages 338–345. IEEE Computer Society.

Courcoubetis, C. and Yannakakis, M. (1995). The complexity of probabilistic verification. *J. ACM*, 42(4):857–907.

David, A., Jensen, P. G., Larsen, K. G., Mikucionis, M., and Taankvist, J. H. (2015). Uppaal stratego. In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 206–211. Springer.

Dräger, K., Forejt, V., Kwiatkowska, M. Z., Parker, D., and Ujma, M. (2015). Permissive controller synthesis for probabilistic systems. *Log. Methods Comput. Sci.*, 11(2).

Eliyahu, T., Kazak, Y., Katz, G., and Schapira, M. (2021). Verifying learning-augmented systems. In *SIGCOMM*, pages 305–318. ACM.

García, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.*, 16:1437–1480.

Gross, D., Jansen, N., Junges, S., and Pérez, G. A. (2022). COOL-MC: A comprehensive tool for reinforcement learning and model checking. In *SETTA*, volume 13649 of *Lecture Notes in Computer Science*, pages 41–49. Springer.

Gross, D., Schmidl, C., Jansen, N., and Pérez, G. A. (2023a). Model checking for adversarial multi-agent reinforcement learning with reactive defense methods. In *ICAPS*, pages 162–170. AAAI Press.

Gross, D., Schmidl, C., Jansen, N., and Pérez, G. A. (2023b). Model checking for adversarial multi-agent reinforcement learning with reactive defense methods. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 162–170.

Hahn, E. M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., and Wojtczak, D. (2019). Omega-regular objectives in model-free reinforcement learning. In *TACAS (1)*, volume 11427 of *Lecture Notes in Computer Science*, pages 395–412. Springer.

Hansson, H. and Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal Aspects Comput.*, 6(5):512–535.

Hasanbeig, M., Kroening, D., and Abate, A. (2020). Deep

reinforcement learning with temporal logics. In *FORMATS*, volume 12288 of *Lecture Notes in Computer Science*, pages 1–22. Springer.

Hensel, C., Junges, S., Katoen, J., Quatmann, T., and Volk, M. (2022). The probabilistic model checker Storm. *Int. J. Softw. Tools Technol. Transf.*, 24(4):589–610.

Jin, P., Wang, Y., and Zhang, M. (2022). Efficient LTL model checking of deep reinforcement learning systems using policy extraction. In *SEKE*, pages 357–362. KSI Research Inc.

Kazak, Y., Barrett, C. W., Katz, G., and Schapira, M. (2019). Verifying deep-rl-driven systems. In *NetAI@SIGCOMM*, pages 83–89. ACM.

Kwiatkowska, M. Z., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer.

Littman, M. L., Topcu, U., Fu, J., Isbell, C., Wen, M., and MacGlashan, J. (2017). Environment-independent task specifications via GLTL. *CoRR, abs/1704.04341*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013a). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013b). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Vamplew, P., Smith, B. J., Källström, J., de Oliveira Ramos, G., Radulescu, R., Roijers, D. M., Hayes, C. F., Heintz, F., Mannion, P., Libin, P. J. K., Dazeley, R., and Foale, C. (2022). Scalar reward is not enough: a response to silver, singh, precup and sutton (2021). *Auton. Agents Multi Agent Syst.*, 36(2):41.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gülçehre, Ç., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354.

Vouros, G. A. (2023). Explainable deep reinforcement learning: State of the art and challenges. *ACM Comput. Surv.*, 55(5):92:1–92:39.

Wang, Y., Roohi, N., West, M., Viswanathan, M., and Dullerud, G. E. (2020). Statistically model checking PCTL specifications on markov decision processes via reinforcement learning. In *CDC*, pages 1392–1397. IEEE.

Zhao, C., Deng, C., Liu, Z., Zhang, J., Wu, Y., Wang, Y., and Yi, X. (2023). Interpretable reinforcement learning of behavior trees. In *ICMLC*, pages 492–499. ACM.

Zhu, H., Xiong, Z., Magill, S., and Jagannathan, S. (2019). An inductive synthesis framework for verifiable reinforcement learning. In *PLDI*, pages 686–701. ACM.