Rule-Guided Reinforcement Learning Policy Evaluation and Improvement

Martin Tappler[1] , **Ignacio D. Lopez-Miguel**[1] , **Sebastian Tschiatschek**[2] and **Ezio Bartocci**[1]
[1]TU Wien
[2]University of Vienna
{martin.tappler, ignacio.lopez, ezio.bartocci}@tuwien.ac.at, sebastian.tschiatschek@univie.ac.at

Abstract

We consider the challenging problem of using domain knowledge to improve deep reinforcement learning policies. To this end, we propose LEGIBLE, a novel approach, following a multi-step process, which starts by mining rules from a deep RL policy, constituting a partially symbolic representation. These rules describe which decisions the RL policy makes and which it avoids making. In the second step, we generalize the mined rules using domain knowledge expressed as metamorphic relations. We adapt these relations from software testing to RL to specify expected changes of actions in response to changes in observations. The third step is evaluating generalized rules to determine which generalizations improve performance when enforced. These improvements show weaknesses in the policy, where it has not learned the general rules and thus can be improved by rule guidance. LEGIBLE supported by metamorphic relations provides a principled way of expressing and enforcing domain knowledge about RL environments. We show the efficacy of our approach by demonstrating that it effectively finds weaknesses, accompanied by explanations of these weaknesses, in eleven RL environments and by showcasing that guiding policy execution with rules improves performance w.r.t. gained reward.

1 Introduction

Deep reinforcement learning (RL) has shown impressive feats. Starting from deep Q-learning [Mnih *et al.*, 2015], where a single network architecture successfully played dozens of Atari games, deep RL has moved on to more complex domains and achieved super-human performance in games, like Go [Silver *et al.*, 2016] and chess [Silver *et al.*, 2018], and it mastered StarCraft [Oriol Vinyals et al., 2019]. Despite the impressive success in winning strategy games, deploying deep RL agents faces challenges in other domains. First, the decision-making of RL agents needs to be more accurate, making it hard to trust their decisions. Second, successful applications are often not the result of RL alone.

A thorough analysis of an agent's behavior may help understand its "reasoning" – e.g., chess masters found that AlphaZero highly values king mobility [Nielsen, 2019]. Explainable AI (XAI) methods strive to alleviate the analysis of policies by providing human-understandable explanations of an RL agent's decision. Relating to the second point, AlphaZero combines RL with Monte Carlo tree search (MCTS) to better evaluate situations. Thus, it considers the application domain, as MCTS works well for decision-making in board games [Chaslot *et al.*, 2008].

Contributions. We propose **poLicy Evaluation GuIded By ruLEs (LEGIBLE)**, a rule-based framework to create explanations of an RL policy's decisions and to evaluate and improve the RL policy under consideration, supported by domain knowledge. LEGIBLE comprises three main methods that build upon each other. **1. Mining Rules:** We first mine rules that partially represent the policy. These rules are split into positive, denoting action choices, and negative rules, denoting action avoidance in particular situations. They enable human comprehension of a policy's decisions and symbolic reasoning and manipulation. **2. Generalizing Rules:** We propose an approach to generalize mined rules to other situations through domain knowledge about symmetries and relations known about the environments. Borrowing the concept from software testing, we formalize the generalization via metamorphic relations [Chen *et al.*, 2018]. **3. Rule-Guided Execution:** Finally, we propose to guide the execution of a policy in the RL environment with generalized rules, i.e., enforce the decisions prescribed by rules, which serves two purposes. **3.1. Evaluation:** Rule-guided execution helps to identify weaknesses in the policy. If we find that enforcing generalized rules corresponding to a certain rule r improves the RL agent's performance, we can deduce that the original rule r is likely adequate. However, the policy has not learned to decide adequately in related situations, i.e., we identify a weakness in the policy's generalization. Additionally, the generalized rules explain the cause of the weakness. **3.2. Policy Improvement:** Finally, guiding policy execution with the composition of generalized rules, which the evaluation deemed useful, creates a new policy that improves upon the original policy. We demonstrate these aspects of LEGIBLE in experiments with policies trained in six PAC-Man RL environments [DeNero and Klein, 2010] and five environments from Farama's `highway-env` [Leurent, 2018].

Related Work. Our work is related to explainable RL (XRL), runtime enforcement, and evaluation of RL policies. [Milani *et al.*, 2024] provide an excellent survey on XRL including a taxonomy and evaluation criteria. Learning rules falls into the most popular taxonomic category of *feature importance* and the subcategory *Convert Policy to an Interpretable Format*. Although symbolic rules as an interpretable format are not popular yet, decision trees which are related to rules are often used [Bewley and Lawry, 2021; Guo and Wei, 2022; Bastani *et al.*, 2018; Milani *et al.*, 2022]. Several neuro-symbolic approaches have been proposed for RL, where neural networks encode symbolic relations and logical rules. Examples include neural logic machines [Dong *et al.*, 2019] and neural logic RL [Jiang and Luo, 2019], and relational approaches [Delfosse *et al.*, 2023; Zambaldi *et al.*, 2018] that enable the extraction of symbolic information and rules from learned policies. In contrast to these works, we consider standard architectures used in deep RL and extract partial rule-based representations of learned policies. Moreover, we demonstrate the application of rules beyond explanations for evaluation and runtime enforcement.

There are two main strands of work in RL policy evaluation: off-policy evaluation (OPE) and testing of RL policies. OPE [Uehara *et al.*, 2022; Chandak *et al.*, 2021; Jiang and Li, 2016] estimates the expected performance of a new policy using existing data from a previously learned policy. Since we generate new data, our approach to evaluation is closer to RL testing [Tappler *et al.*, 2022; Tappler *et al.*, 2024b; Tappler *et al.*, 2024a; Zolfagharian *et al.*, 2023; Li *et al.*, 2023; Biagiola and Tonella, 2024], which creates challenging situations to test policies in them. These approaches apply software testing concepts, like search-based testing, to RL. To our knowledge, we are the first to apply metamorphic testing in RL, which is popular for testing other machine-learning models [Zhang *et al.*, 2022; Xie *et al.*, 2011; Guo *et al.*, 2020] due to its applicability when absolute correctness criteria are not available. To test policies, we perform rule-guided execution of policies, which can be considered a type of runtime enforcement [Falcone, 2010]. In RL, runtime enforcement has gained popularity in the form of *shielding* [Alshiekh *et al.*, 2018; Odriozola-Olalde *et al.*, 2023], which enforces pre-specified properties like safety. In contrast, we enforce generalizations of learned rules to evaluate these generalizations.

2 Preliminaries

2.1 Reinforcement Learning

An RL agent learns a decision-making policy for a task by trial and error. At each step, the agent observes the environment's state and performs an action, triggering a stochastic state transition. It then receives feedback in the form of a numerical value, called reward, telling it how well it is doing, and the new state of the environment. During training, the agent learns how to maximize the cumulative reward it gets.

Let $\Delta(S)$ denote a probability distribution over a set S. Formally, an agent interacts with a Markov decision process (MDP) $\mathcal{M} = \langle \mathcal{S}, s_0, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ consisting of a set of states $\mathcal{S}$, an initial state $s_0 \in \mathcal{S}$, a set of actions $\mathcal{A}$, a probabilistic transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, and a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Delta(\mathbb{R})$. The agent's behavior is characterized by a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, defining a distribution over actions to take in given states. Executing a policy π within an environment modeled by an MDP $\mathcal{M}$ yields traces of the form $s_0, a_0, r_1, s_1, \ldots, a_{n-1}, r_n, s_n$ with $r_i \sim \mathcal{R}(s_{i-1}, a_{i-1}, s_i)$, $\mathcal{P}(s_i, a_i)(s_{i+1}) > 0$, and $\pi(s_i)(a_i) > 0$. Such a finite execution is also called an episode. The agent's goal is to learn a policy that maximizes the discounted cumulative reward $R = \mathbb{E}(\sum_{i=0}^{\infty} \gamma^i r_{i+1})$ for a discount factor $\gamma \in [0, 1)$.

To handle large state spaces, RL algorithms often employ neural networks as function approximators, e.g., for state-action value functions [Mnih *et al.*, 2015]. Since we mostly focus on extracting symbolic knowledge from policies in this paper, we do not detail the specifics of RL algorithms.

State-Action Value Function. Q-learning [Watkins and Dayan, 1992] and its deep variants [Mnih *et al.*, 2015; van Hasselt *et al.*, 2016] are based on the notion of Q-function, or state-action value function. This function is defined as $Q_\pi(s, a) = \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_{i+1} | s_0 = s, a_0 = a]$, i.e., it is the expected return after executing a in state s and then following policy π. We focus mainly on Q-learning agents and use the Q-function to identify actions that an agent avoids in a given situation, i.e., actions with low Q-values.

2.2 Explainable & Interpretable AI

Explainable AI (XAI) methods create human-understandable explanations of individual decisions or predictions of models that are otherwise not interpretable, like neural networks, while interpretability techniques commonly extract useful information, like interpretable surrogate models, from non-interpretable models [Molnar, 2022]. To enable human comprehension, explanations commonly focus on the most relevant factors leading to a decision, like the most important input features. This working principle makes XAI techniques good candidates for creating abstractions for symbolic AI. We use a model-agnostic XAI technique for explaining individual predictions, called LIME [Ribeiro *et al.*, 2016], and we use rules as (partial) surrogate models of RL policies.

LIME. Treating the choice of action a through a policy π in state s as a classification task, LIME can learn a local interpretable surrogate model around s. It samples new data points around s and queries π to learn the surrogate model, which explains what features of s lead to the choice of a and provides weights describing the strength of influence. As we consider states that are vectors containing information about the environment, we use LIME for tabular data. In this case, it samples new values in the neighborhood of numerical features and values from a *training set* for categorical features.

We use the Python implementation of LIME [Marco Tulio Ribeiro et al., 2021], which provides an explanation for predicting every available class, i.e., for choosing every available action. Several alternatives to LIME exist, like SHAP [Lundberg and Lee, 2017], but we have chosen LIME as it is reasonably fast and showed promising results in experiments. Since it is model-agnostic and works for image and textual data, LIME enables us to easily adapt to changing representations of π and to handle additional types of features. **Interpretability & Rules.** RL interpretability is approached

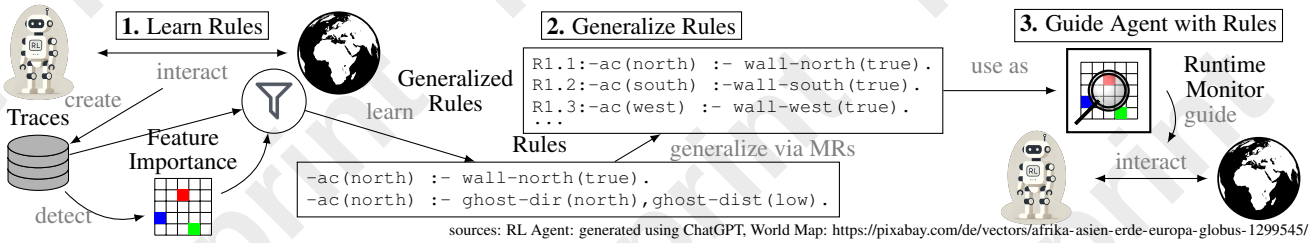

Figure 1: Overview of the proposed approach

in different ways, e.g., post-hoc interpretability for a DRL policy π may be achieved by learning a decision tree policy from π through imitation learning [Bastani *et al.*, 2018]. We learn rule-based representations of policies, which, like decision trees, enable manual comprehension and symbolic reasoning [Apté and Weiss, 1997]. The rules are of the form $a \leftarrow c_1, \ldots, c_n$, specifying to take action a if conditions c_1 to c_n hold. We have chosen rules, as they conveniently enable learning partial representation of a policy. For rule learning, we use the algorithm RIPPER [Cohen, 1995], where we treat the selection of actions as a classification task from states to actions. To clearly distinguish between reinforcement learning and rule learning, we refer to the latter as rule mining.

2.3 Metamorphic Testing

Metamorphic testing (MT) [Chen *et al.*, 2018] is a technique for generating test cases and deciding on test verdicts. It is based on metamorphic relations (MRs), which define a relation between a sequence of a program's inputs and the corresponding outputs. MRs express the output changes in response to input changes. Consider, e.g., a program implementing the factorial, an MR could be defined as $\langle (n, n+1), (r, r \cdot (n+1)) \rangle$, where the first pair represents two inputs and the second pair represents two outputs denoting that if $n! = r$ then $(n+1)! = r \cdot (n+1)$ should hold. Note that MRs do not specify correctness criteria in absolute terms but as relations. This makes MT a popular choice for deciding on test verdicts in machine learning [Zhang *et al.*, 2022], e.g., in image recognition [Dwarakanath *et al.*, 2018]. While it is hardly possible to completely characterize a cat, MRs can express properties like *if an image shows a cat, then a rotated version of that image still shows a cat*. In this paper, we adapt MRs from programs to RL agents, by forming relations over states (inputs to the agent) and actions (outputs of the agent) to express domain knowledge. Unlike in MT, we do not make assumptions about the correctness of specific chosen actions.

3 LEGIBLE

This section presents an overview of **poLicy Evaluation GuIded By ruLEs (LEGIBLE)**, our rule-based framework for policy evaluation and improvement, comprising the three main steps depicted in Fig. 1. Step 1 learns symbolic rules that (partially) capture the decision-making of a given deep RL policy. Rules are Horn clauses, where the head specifies an action and the body defines states in which the rule should be applied. We distinguish positive rules, for situations where the policy chooses a certain action, and negative rules describing situations where it avoids a certain action. Rule mining applies two types of criteria: (1) the rules should reflect what features are important to the decisions of the RL policy, and (2) the rules should be accurate and cover as many situations as possible, where we favor accuracy. Hence, mined rules explain which decisions are important and occur often.

Step 2 generalizes rules to new situations that are related to the originally covered situations through user-specified relations. For this purpose, we use metamorphic relations (MRs) [Chen *et al.*, 2018] from software testing, which usually specify how program outputs should change in response to input changes. Likewise, we use them to specify how decisions should change in response to a change in the state. MRs generally need to be created manually and they reflect some domain knowledge about the environment, like symmetry constraints, i.e., through Step 2, we provide a method to introduce symbolic domain knowledge into RL.

Finally, in Step 3 we apply the generalized rules during executions of the RL policy under consideration. Actions prescribed by rules generalized from positive rules are enforced while actions of negative generalized rules are blocked. In this way, we determine which rules generalize to improved behavior in the RL environment. Conversely, such improvements reveal weaknesses in the RL policy since they show that the policy makes useful decisions in some specific situations but it does not generalize these decisions to all related situations. Hence, Step 3 provides a way to evaluate RL policies and a basis for policy improvements by enforcing sets of generalized rules that influence behavior positively.

Setting. For the remainder of this paper, let π be the greedy policy of a Q-learning-based agent and $Q\colon S \times A \to \mathbb{R}$ be its Q-function. A state s is a vector in $\mathbb{R}^n$, where n is the number of features, f_i for $i \in [1..n]$ denotes the i^{th} feature, and f_i^s denotes its value in s. To enable rule learning, we consider environments with a discrete action space and discretize the state space by defining intervals for every feature separately. To simplify the presentation, we write $f_i^s = k$ to denote $f_i^s \in [l_{i,k}, u_{i,k}]$ where $[l_{i,k}, u_{i,k}]$ is the k^{th} interval for feature i.

Running Example. Figure 2 shows a small PAC-Man environment from [DeNero and Klein, 2010], which serves as a running example. The agent can move in the four cardinal directions or do nothing at every time step. Its goal is to eat all pellets (small dots). Colliding with ghosts terminates an episode unless they are vincible, which happens for a fixed amount of time after the agent eats a capsule (large dot). The observable states include information on the location of walls,

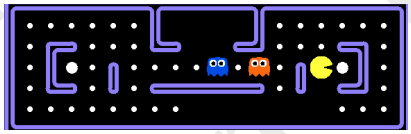

Figure 2: Small PAC-Man environment

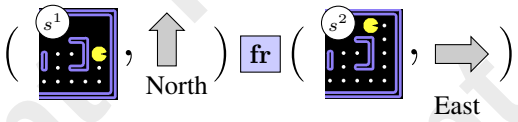

Figure 3: State-action pairs related by $\mathbf{fr} = \langle 0, 2, 62, 64, = \rangle$.

the distance to ghosts, capsules, and pellets, the direction towards these objects, and other data. For efficient learning, we use a one-hot encoding for categorical features, e.g., features 9 to 12 describe the direction toward the closest pellet, where $f_9^s = 1$ means that the closest pellet is north.

The agent receives a reward of 10 for eating a pellet, 200 for eating a ghost (collision when vincible), 500 for completing the level, -500 for colliding with an invincible ghost, and -1 every time step to encourage fast completion.

4 Step 1: Rule Mining

To create training data for rule mining, we execute the agent in its environment for n_{rule} episodes to sample traces, i.e., state-action-reward sequences. From these traces, we collect all observed state-action pairs $(s, a) \in S \times A$ in a multiset E to which we refer as experiences. We treat rule mining as a classification problem from states to actions based on training data E, where we aim to learn rules that (1) represent what is important to the agent's decisions and that (2) are accurate and have high coverage. In the following, we detail the form of rules we consider and how we learn them.

Rules. A rule ρ is of the form:

$$\rho = \otimes action(a) \leftarrow \bigwedge_{i \in I} f_i = v_i, \text{ where} \qquad (1)$$

$\otimes \in \{+, -\}$ is the rule polarity, and $I \subseteq [1..n]$ is a set of (feature) indices. We refer to the conjunction of conditions as the rule body, denoted $body(\rho)$, and to the action consequent as the rule head. Given a state s which satisfies $\bigwedge_{i \in I} v_i = f_i^s$, we say that ρ triggers in s, denoted $s \models \rho$. If $\otimes = -$, ρ is a negative rule, denoting action a is avoided if $s \models \rho$. A positive rule with $\otimes = +$ denotes that a should be taken.

Feature Importance. To determine which features are important to the policy's decision, we select n_{feat} experiences from E and apply LIME to generate explanations for the decisions of π. The explanations quantify the importance of a feature f_i to perform or not perform an action a. For each f_i, we compute the sum of importance values from the n_{feat} explanations, which we denote by $imp(\otimes, a, f_i)$, representing the importance of feature f_i to take or avoid a.

Mining Rules. We apply the rule learning algorithm RIPPER [Cohen, 1995] for every combination of action and rule polarity individually. This approach enables more effective mining of negative rules compared to posing rule mining as a multi-class classification problem for all actions. Hence, we select two datasets from E for every action-polarity combination, which rules shall distinguish: the inclusion dataset $inc(a, \otimes)$ containing positive examples and the exclusion dataset $exc(a, \otimes)$. To account for feature importance, we adapt RIPPER's rule growing. Instead of solely targeting coverage by optimizing FOIL's information gain during rule growing, we maximize the product of FOIL's information gain and feature importance $imp(\otimes, a, f_i)$.

The inclusion dataset $inc(a, +)$ for mining positive rules contains all (s, a) from E, i.e., experience matching action a, and the exclusion dataset contains all (s, a') from E s.t. $a' \neq a$. To mine negative rules, we add all (s, a') where $a' = \arg\min_a Q(s, a)$ in $inc(a, +)$ and we add all (s, a) to $exc(a, -)$. Following the rule mining through RIPPER, we filter rules by imposing bounds on minimal accuracy and coverage. We evaluate the accuracy of a rule by checking if it agrees with the experiences from a validation set. Coverage refers to the ratio of times that a rule triggers, which is a common optimization criterion of rule learning algorithms.

Running Example. In the PAC-Man environment, we learn rules, such as $\rho_1 = -action(0) \leftarrow f_{62} = 0$ and $\rho_2 = +action(0) \leftarrow f_9 = 1 \wedge f_{50} = 1 \wedge f_{62} = 1 \wedge f_{65} = 1$. Both predicate on action 0, i.e., going north. Rule ρ_1 says that PAC-Man avoids trying to go north if there is a wall, as $f_{62} = 0$ means that going north is not possible. The second rule ρ_2 says that the agent learned to go north if there is a pellet in the north direction ($f_9 = 1$), Ghost 2 is west ($f_{50} = 1$), and there is no wall to the north ($f_{62} = 1$) and west ($f_{65} = 1$).

5 Step 2: Rule Generalization

The rules provide insights into decisions that the policy learned to be beneficial. The intuition behind rule generalization is that often there are symmetries in the environment and the agent may have learned how to act in one situation, but not in corresponding symmetrical situations. More generally, situations are often related through relations that can be expressed symbolically. Borrowing the concept from software testing, we refer to these symbolic relations as metamorphic relations (MRs). We define MRs for RL based on feature relations that relate actions and individual features of states.

Definition 1. *A* feature relation **fr** *is a tuple* $\langle a, a', i, j, \mathcal{R} \rangle$ *with* $\mathcal{R} \subseteq \mathbb{R} \times \mathbb{R}$ *describing a relation between features* i *and* j *in state-action pairs. Two state-action pairs* (s_1, a_1) *and* (s_2, a_2) *are feature-related by* **fr** *denoted* (s_1, a_1) **fr** (s_2, a_2) *iff* $a_1 = a \wedge a_2 = a' \wedge (f_i^{s_1}, f_j^{s_2}) \in \mathcal{R}$. *A* metamorphic relation **mr** *for RL is a set of feature relations defined on the same actions* a *and* a'. *Given* $sa_1 = (s_1, a_1)$ *and* $sa_2 = (s_2, a_2)$, *we define* sa_1 **mr** sa_2 *iff* sa_1 **fr** sa_2 *for all* **fr** $\in$ **mr**, *i.e., an MR relates multiple pairs of features of state-action pairs.*

Running Example. Figure 3 shows two PAC-Man state-action pairs related by $\langle 0, 2, 62, 64, = \rangle$, where $=$ simply denotes the equality relation. On the left, PAC-Man goes north action with action 0 and there is no wall to the north, represented by $f_{62}^{s^1} = 1$. On the right, PAC-Man goes east with action 2 and there is no wall to the east $f_{64}^{s^2} = 1$, i.e., we have $f_{62}^{s^1} = f_{64}^{s^2}$ satisfying the relation from **fr**.

Algorithm 1 Policy evaluation guided by Rules

Input: Q-function Q, set of gen. rules G, # eval. episodes n
Output: Average Cumulative Reward

```
1: Rews ← ⟨⟩
2: for i ← 1 to n do
3:     s ← RESET(), rew ← 0
4:     while s not terminal do
5:         G_t ← {(action(r), polarity(r)) | r ∈ G, s ⊨ r}
6:         if |{(a, +) ∈ G_t}| = 1 then
7:             act ← a
8:         else
9:             q ← Q(s, ·)
10:            q(s, a) ← −∞ for (a, −) ∈ G_t
11:            act ← arg max_a q(s, a)
12:        s, r ← STEP(s, act), rew ← rew + r
13:    APPEND(Rews, rew)
14: return mean(Rews), stderr(Rews)
```

We define MRs based on feature relations to enable convenient, compositional definitions and extend them to rules as follows. Let $\mathbf{fr} = \langle a, a', j, k, \mathcal{R} \rangle$, $\rho_1 = \otimes_1 action(a_1) \leftarrow \bigwedge_{i \in I_1} f_i = v_{i1}$, and $\rho_2 = \otimes_2 action(a_2) \leftarrow \bigwedge_{i \in I_2} f_i = v_{i2}$: ρ_1 **fr** ρ_2 iff $\otimes_1 = \otimes_2$, $a_1 = a$, $a_2 = a'$, and either $f_j = x \in body(\rho_1)$ and $f_k = y \in body(\rho_2)$ such that $(x, y) \in \mathcal{R}$ or $j \notin I_1$ and $k \notin I_2$. Rule bodies must either include facts satisfying the relation $\mathcal{R}$, or they should not predicate on the respective features. Analogously to states, we extend MRs to rules denoted by ρ_1 **mr** ρ_2. Through sets of MRs M, we automatically generalize each positive and negative rule ρ_j to a set of positive or negative rules given by the fixed point of $M_{gen}(\rho_j) = \{\rho_j\} \cup \bigcup \{M_{gen}(\rho) \mid \rho_j \ \mathbf{mr}\ \rho, \mathbf{mr} \in M\}$.

Running Example. We illustrate rule generalization with rule $\rho_1 = -action(0) \leftarrow f_{62} = 0$ and feature relation $\mathbf{fr} = \langle 0, 2, 62, 64, = \rangle$ from above. An MR $\mathbf{mr} = \{\mathbf{fr}\}$ including only **fr** describes a 90-degree clockwise rotation of the action and the only relevant feature from north to east. Action 0 and 2 describe going north and east, respectively. Features 62 and 64 indicate the absence of a wall to the north and east, respectively, e.g., $f_{64} = 0$ means that there is a wall to the east. Hence, if $\rho_2 = -action(2) \leftarrow f_{64} = 0$ then ρ_1 **mr** ρ_2, because ρ_1 **fr** ρ_2, which holds because $polarity(\rho_1) = polarity(\rho_2)$, $head(\rho_1) = 0$, $head(\rho_2) = 2$, and $f_{62} = 0 \in body(\rho_1)$, $f_{64} = 0 \in body(\rho_2)$ with $0 = 0$. The rule ρ_2 can be automatically generated and states to not go east if there is a wall. Enforcing it may improve performance if the agent has not learned to generalize from ρ_1 to ρ_2, because trying to go into a wall results in a reward of -1 as the agent stays in its location in such a case.

6 Step 3: Guiding Agents with Rules

To evaluate π, we propose to first execute π without rule guidance and then guide π with each rule set $M_{gen}(\rho_j)$ separately for a set M of MRs. By comparing the gained cumulative reward, we determine which generalized rules improve performance and thus reveal weaknesses of π.

Algorithm 1 monitors the execution of π for n episodes and enforces rules if they trigger. It assumes that the environment provides RESET and STEP operations, which are commonly part of the interface to RL environments, like the Gymnasium API [Towers *et al.*, 2024]. After resetting the environment to start an episode, in every step, Line 5 checks which rules trigger. If a single positive rule triggers (Line 6), we enforce it. Otherwise – there is no positive rule or a conflict between positive rules – we disable all actions of negative rules in Line 10 by setting their Q-values to negative infinity. After that, we choose the action with the highest Q-value or a random action if all actions have been disabled. Furthermore, we could add additional randomness to the action choices. Finally, Algorithm 1 returns the average cumulative reward and the corresponding standard error. While generalization does not change rule polarity, Algorithm 1 supports combining rules generalized from rules with different polarity.

7 Experiments

This section presents experiments on the application of learned and generalized rules. First, we show how generalized rules help detect weaknesses in RL policies through rule-guided execution and how rules can explain the identified weaknesses. After that, we demonstrate the RL policy improvement through rule-guided execution.

Setup and Environment. All experiments are based on RL policies trained in six PAC-Man levels [DeNero and Klein, 2010] and five `highway-env` [Leurent, 2018] environments using stable-baselines3 [Raffin *et al.*, 2021]. The PAC-Man levels differ in size (small, medium, and original) and the presence of capsules. We trained DQN [Mnih *et al.*, 2015] policies for $2.5 \cdot 10^6$ steps in the small and medium PAC-Man environments with 69-dimensional states and for $5 \cdot 10^6$ steps in the original-sized environments with 117-dimensional states. In `highway-env`, we trained DQN policies for $5 \cdot 10^5$ steps to navigate in driving scenarios, like merging onto a highway. We configured `highway-env` to create observations relative to the ego vehicle, comprising 7 properties of the ten closest vehicles, like their relative positions. Every training run is repeated five times and the resulting policies provide the basis for the experiments below. Code and data from the experiments are available at https://doi.org/10.6084/m9.figshare.28569017.

We mine rules for all environments using the same setup, except for the discretization of states. In the PAC-Man environments, we use the `decile` discretization provided by LIME [Marco Tulio Ribeiro et al., 2021], which discretizes each numerical feature into intervals corresponding to deciles calculated from the data. Categorical features, like cardinal directions to the closest food, are left unchanged. The `highway-env` environments do not include categorical features and there we normalize and discretize each feature into ten intervals. For both types of environments, we sample $n_{rule} = 600$ episodes to generate experiences for rule mining and impose a minimal accuracy of 0.9 and a minimal coverage of 0.01 on rules, discarding all other rules.

For the remainder of this section, let π denote a trained policy, R be a set of rules learned from π, and M be a set of MRs. The MRs for PAC-Man encode 90-degree clockwise turns from every cardinal direction, i.e., from every rule, we generate four rules, including the original rule. The

Experiment	Detected Weaknesses			# Evaluations
	RT	RR	MT (Ours)	
small	0.00	0.01	**0.13**	105.4
small-nc	0.01	0.03	**0.11**	85.0
medium	0.01	0.04	**0.08**	117.6
medium-nc	0.01	0.00	**0.09**	109.6
original	0.00	0.00	**0.16**	121.6
original-nc	0.02	0.03	**0.14**	108.8

Table 1: Detected weaknesses in six PAC-Man levels.

Experiment	Detected Weaknesses			# Evaluations
	RT	RR	MT (Ours)	
highway	0.02	**0.14**	0.05	90.8
highway-fast	0.07	**0.20**	0.12	119.6
merge	0.09	**0.24**	0.16	109.6
intersection	0.06	0.00	**0.12**	20.8
roundabout	0.01	0.06	**0.11**	33.6

Table 2: Detected weaknesses in `highway-env` environments.

`highway-env` MRs generalize rules to other vehicles occurring in observations, i.e., from a rule with conditions on the first vehicle, we generate rules conditioned on the other vehicles. Furthermore, the MRs encode symmetry constraints, e.g. generalizing rules for changing to the left lane to rules for changing to the right lane and vice versa.

7.1 Identifying Weaknesses

With the first set of experiments, we approach the research question RQ1: *Does rule-guided execution effectively reveal weaknesses in a trained policy?* We consider policy weaknesses as decisions where enforcing a rule improves performance, i.e., rule-guided execution detects decisions that could be improved. As a benchmark for performance, we compare against the average cumulative reward cr_π gained by π without rule guidance. To identify weaknesses in π, we propose to generalize each rule $r \in R$ to $R_g = M_{gen}(r)$ individually and guide the execution of π with R_g using Algorithm 1. We perform $n = 100$ evaluation episodes in `highway-env` and $n = 250$ evaluation episodes in the PAC-Man environments. R_g reveals a weakness if the cumulative rewards from rule-guided execution are significantly larger than those of unguided execution, which we determine using a Welch test [Welch, 1947] and a p-value threshold of 0.05.

Baselines. We are the first to propose MT for RL, therefore we compare MT to two random baselines. The first baseline, *random testing (RT)*, blocks or enforces randomly chosen actions during policy execution in k percent of the states. We randomly assign each of the k policy changes to approximately one percent of the state space, such that if we visit a state twice the same change happens. In the experiments, we performed 100 RT evaluations where we only blocked actions and 100 evaluations where only enforced actions. We set $k = 3$ since we found that low values are more effective at detecting weaknesses. Each evaluation compares against the average cumulative reward cr_π from unguided execution.

Additionally, we use another random baseline, which we call *random rules* (RR). For RR, we randomly generate rule sets instead of generating them via MRs and execute Algorithm 1, i.e., RR serves as an **ablation study** to study the impact of MRs. The random generation uses information extracted from mined rules by creating rules with the same distribution of rule lengths and rule polarities, and by using the feature values found in the mined rules. Additionally, rule sets created for RR are of similar size as the rule sets created using MRs. Hence, the random rules benefit from information extracted from policies, but not from domain knowledge.

Results. Tables 1 and 2 show evaluation results for the five base policies trained in each environment. For each RT, RR, and rule-guided execution supported by MRs, denoted as metamorphic testing (MT), the tables show the mean ratio of evaluations that detected weaknesses from five repetitions. The tables additionally show the number of evaluations for RR and MT (for RT they are fixed to two times 100), i.e., the number of generalized rule sets R_g. We can see that on average 5 to 24 percent of the MT evaluations reveal weaknesses. This is consistently higher than RT, especially in PAC-Man environments, where RT rarely reveals weaknesses. In contrast, RR reveals more weaknesses than MT in three cases where it substantially lees weaknesses in the other eight environments. We can answer RQ1 positively, as rule-guided execution consistently revealed weaknesses in the examined policies, both through RR and MT. We can further deduce that domain knowledge helps since MT performed better overall. In the cases, where RR performed better than MT, either our MRs may not optimally represent relations that hold in the environment or the policy under test generalizes well, but includes other issues that RR detects. MT further improves upon RR as it facilitates explanations, as the rules within a rule set are related. If a rule set R_g reveals weakness, we know that the cause is linked to the original rule from which R_g was generated and to the MRs used for generation. Beyond that, the good performance of RR suggests that search-based approaches, which are popular in RL testing [Zolfagharian *et al.*, 2023; Tappler *et al.*, 2022], might be viable for rule generation.

Explaining Weaknesses. We illustrate explaining policy weaknesses with two cases from the PAC-Man levels *small* and *original* with capsules. In both cases, the generalization of the following simple rules increases performance:

$$-action(0) \leftarrow f_9 = 0 \wedge f_{62} = 0 \qquad \text{(small)}$$
$$-action(0) \leftarrow f_9 = 0 \wedge f_{110} = 0 \qquad \text{(original)}$$

The feature indices differ among the environments, but both rules can be interpreted as "*don't go north if there is no food and there is a wall to the north*". Generalizations to other directions are safe to apply, yet the policies have not picked up on them, thus enforcing them improves performance. However, we found that the policies have learned stronger versions of these rules, e.g., in the environment *original*, we learned related rules with additional constraints: $-action(1) \leftarrow f_2 = 0 \wedge f_9 = 0 \wedge f_{10} = 0 \wedge f_{11} = 1 \wedge f_{88} = 0 \wedge f_{111} = 0$ and $-action(3) \leftarrow f_5 = 0 \wedge f_{10} = 1 \wedge f_{12} = 0 \wedge f_{113} = 0$. Hence, rule-guided execution points to situations where the policy learned suboptimal decisions.

Algorithm 2 Greedy rule selection for policy improvement.

```
cr_max ← mean cumulative reward of π
RT ← ∅
for ρ ∈ R do
    if Alg. 1 with G ← M_gen(ρ) = R_g reveals weakness then
        RT' ← RT ∪ R_g
        cr, stderr ← Algorithm 1 with G ← RT'
        if cr > cr_max then
            RT ← RT', cr_max ← cr
```

Experiment	Base	Rule-Guided (Ours)	Ext. Training	RS
small	1023.6 ±43.7	**1268.9 ±20.5**	1018.4 ±14.0	4.6
small-nc	450.6 ±36.5	**635.7 ±16.8**	447.9 ±11.6	4.4
medium	1339.6 ±49.0	**1601.1 ±22.0**	1371.4 ±14.9	4.4
medium-nc	840.9 ±38.6	**1050.2 ±16.9**	789.4 ±12.4	3.8
original	1590.5 ±51.5	**2013.6 ±28.8**	1552.4 ±16.3	4.2
original-nc	784.8 ±48.3	**1194.8 ±28.1**	796.4 ±15.0	3.4

Table 3: Average cumulative reward and standard error for non-guided and rule-guided execution of PAC-Man.

7.2 Policy Improvements

The remaining experiments focus on policy improvement through rule-guided execution, where we tackle Research Question RQ2: *Can guidance with compositions of rule sets improve RL policies?* For this purpose, we assume to have already evaluated π with all generated sets of rules R_g individually. To improve π, we propose to compose rule sets (RS) that reveal weaknesses without overconstraining the policy. Below, we present experiments, where we first greedily selected rule compositions RT through Algorithm 2.

Table 3 and Table 4 show the mean cumulative reward of non-rule-guided execution and execution guided by RT using Algorithm 1, again averaged over five repetitions. The tables also show the corresponding standard error of the estimate and how many rule sets have been composed to create RT. We performed the non-rule-guided execution with the base policies from which we mined rules and with policies trained twice as long, e.g., policies for `highway-env` were trained for 10^6 steps. We can see that rule-guided execution always improved the base reward substantially, except in the *intersection* environment, where the improvement is only marginal on average. In other cases, we see larger improvements, e.g., 274 percent in the *highway-fast* environment. We can further see that extended training did not improve the reward in many environments (PAC-Man) and only slightly in others (`highway-env`). In all cases, rule-guided execution yields larger improvements, thus it provides value that cannot be gained from more training. Moreover, the greedy selection of rules always composed multiple rule sets to create RT to improve the policy under consideration. Hence, we can answer RQ2 positively.

Experiment	Base	Rule-Guided (Ours)	Ext. Training	RS
highway	41.7 ±3.3	**110.4 ±9.7**	68.9 ±5.1	5.4
highway-fast	48.1 ±4.5	**193.8 ±14.1**	74.3 ±5.2	3.6
merge	23.8 ±0.3	**28.7 ±0.1**	25.6 ±0.3	2.8
intersection	11.5 ±0.7	**12.7 ±0.5**	11.7 ±0.5	2.0
roundabout	361.0 ±45.4	**777.2 ±21.4**	499.0 ±32.9	4.6

Table 4: Average cumulative reward and standard error for non-guided and rule-guided execution of policies in `highway-env`.

8 Conclusion

We propose LEGIBLE, a rule-based framework for evaluating, explaining, and improving RL policies. After learning rules from the most important policy decisions, we use high-level domain knowledge to generalize learned rules to other related situations. For this purpose, we leverage MRs [Chen *et al.*, 2018] to express relations like symmetries that hold in the considered environment. By executing policies guided by generalized rules, we identify weaknesses, where a policy learned to behave adequately in a particular situation, but not in related situations. That is, we find cases where RL policies fail to generalize. In these cases, the generalized rules and applied MRs explain the found weaknesses. LEGIBLE provides a basis for policy improvement, by enforcing sets of generalized rules, which were found to improve performance. In experiments with deep RL policies trained in six PAC-Man environments [DeNero and Klein, 2010] and in five `highway-env` [Leurent, 2018] environments, LEGIBLE revealed weaknesses in all policies, which are explained by the learned and generalized rules. Rule-guided execution improved the average cumulative reward by up to 273%.

LEGIBLE enables the integration of domain knowledge into RL policies and our evaluation approach is the first metamorphic testing (MT) approach for RL. In contrast to existing work on RL testing [Tappler *et al.*, 2022; Zolfagharian *et al.*, 2023; Li *et al.*, 2023; Biagiola and Tonella, 2024], which brings the agent into challenging environment states, we take an agent-centric view, evaluating changes in the agent's decisions. By generalizing decisions learned from a policy, we ensure that generalized decisions are learnable.

In future work, we will investigate how to integrate other types of domain knowledge, e.g., by expressing temporal dependencies via restraining bolts [Giacomo *et al.*, 2020]. As our work is complementary to existing RL testing approaches, we will combine MT with approaches that focus on the environment, like search-based testing [Tappler *et al.*, 2022; Zolfagharian *et al.*, 2023; Biagiola and Tonella, 2024]. Finally, we will work on a seamless integration of generalized rules into RL policies. Specifically, we are exploring how to integrate improvements resulting from rules into DRL policies, without needing to enforce them explicitly. To this end, we consider transfer learning [Torrey and Shavlik, 2010] and imitation learning as a basis [Hussein *et al.*, 2017].

Acknowledgments

The project leading to this application has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101034440. This work has further been funded by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT22-023], [10.47379/ICT19018], [10.47379/ICT20058].

References

[Alshiekh *et al.*, 2018] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *AAAI*, pages 2669–2678. AAAI Press, 2018.

[Apté and Weiss, 1997] Chidanand Apté and Sholom M. Weiss. Data mining with decision trees and decision rules. *Future Gener. Comput. Syst.*, 13(2-3):197–210, 1997.

[Bastani *et al.*, 2018] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *NeurIPS*, pages 2499–2509, 2018.

[Bewley and Lawry, 2021] Tom Bewley and Jonathan Lawry. Tripletree: A versatile interpretable representation of black box agents and their environments. In *AAAI*, pages 11415–11422. AAAI Press, 2021.

[Biagiola and Tonella, 2024] Matteo Biagiola and Paolo Tonella. Testing of deep reinforcement learning agents with surrogate models. *ACM Trans. Softw. Eng. Methodol.*, 33(3):73:1–73:33, 2024.

[Chandak *et al.*, 2021] Yash Chandak, Scott Niekum, Bruno C. da Silva, Erik G. Learned-Miller, Emma Brunskill, and Philip S. Thomas. Universal off-policy evaluation. In *NeurIPS*, pages 27475–27490, 2021.

[Chaslot *et al.*, 2008] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game AI. In *AIIDE*. The AAAI Press, 2008.

[Chen *et al.*, 2018] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Comput. Surv.*, 51(1):4:1–4:27, 2018.

[Cohen, 1995] William W. Cohen. Fast effective rule induction. In *ICML*, pages 115–123. Morgan Kaufmann, 1995.

[Delfosse *et al.*, 2023] Quentin Delfosse, Hikaru Shindo, Devendra Singh Dhami, and Kristian Kersting. Interpretable and explainable logical policies via neurally guided symbolic abstraction. In *NeurIPS*, 2023.

[DeNero and Klein, 2010] John DeNero and Dan Klein. Teaching introductory artificial intelligence with Pac-Man. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(3):1885–1889, July 2010.

[Dong *et al.*, 2019] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *ICLR*. OpenReview.net, 2019.

[Dwarakanath *et al.*, 2018] Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M. Rao, R. P. Jagadeesh Chandra Bose, Neville Dubash, and Sanjay Podder. Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In *ISSTA*, pages 118–128. ACM, 2018.

[Falcone, 2010] Yliès Falcone. You should better enforce than verify. In *RV*, volume 6418 of *LNCS*, pages 89–105. Springer, 2010.

[Giacomo *et al.*, 2020] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Restraining bolts for reinforcement learning agents. In *AAAI*, pages 13659–13662. AAAI Press, 2020.

[Guo and Wei, 2022] Wei Guo and Peng Wei. Explainable deep reinforcement learning for aircraft separation assurance. In *DASC*, pages 1–10, 2022.

[Guo *et al.*, 2020] Qianyu Guo, Xiaofei Xie, Yi Li, Xiaoyu Zhang, Yang Liu, Xiaohong Li, and Chao Shen. Audee: Automated testing for deep learning frameworks. In *ASE*, pages 486–498. IEEE, 2020.

[Hussein *et al.*, 2017] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2):21:1–21:35, 2017.

[Jiang and Li, 2016] Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 652–661. JMLR.org, 2016.

[Jiang and Luo, 2019] Zhengyao Jiang and Shan Luo. Neural logic reinforcement learning. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 3110–3119. PMLR, 2019.

[Leurent, 2018] Edouard Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018.

[Li *et al.*, 2023] Zhuo Li, Xiongfei Wu, Derui Zhu, Mingfei Cheng, Siyuan Chen, Fuyuan Zhang, Xiaofei Xie, Lei Ma, and Jianjun Zhao. Generative model-based testing on decision-making policies. In *ASE*, pages 243–254. IEEE, 2023.

[Lundberg and Lee, 2017] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NeurIPS*, pages 4765–4774, 2017.

[Marco Tulio Ribeiro et al., 2021] Marco Tulio Ribeiro et al. Python implementation of LIME, 2021. Available at GitHub: https://github.com/marcotcr/lime.

[Milani *et al.*, 2022] Stephanie Milani, Zhicheng Zhang, Nicholay Topin, Zheyuan Ryan Shi, Charles A. Kamhoua, Evangelos E. Papalexakis, and Fei Fang. MAVIPER: learning decision tree policies for interpretable multi-agent reinforcement learning. In *ECML PKDD*, volume 13716 of *LNCS*, pages 251–266. Springer, 2022.

[Milani *et al.*, 2024] Stephanie Milani, Nicholay Topin, Manuela Veloso, and Fei Fang. Explainable reinforcement learning: A survey and comparative review. *ACM Comput. Surv.*, 56(7):168:1–168:36, 2024.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

[Molnar, 2022] Christoph Molnar. *Interpretable Machine Learning*. 2 edition, 2022.

[Nielsen, 2019] Peter Heine Nielsen. The exciting impact of a game changer: When Magnus met AlphaZero, 2019. Appeared in New in Chess and published online at https://www.newinchess.com/media/wysiwyg/product_pdf/872.pdf.

[Odriozola-Olalde *et al.*, 2023] Haritz Odriozola-Olalde, Maider Zamalloa, and Nestor Arana-Arexolaleiba. Shielded reinforcement learning: A review of reactive methods for safe learning. In *SII*, pages 1–8. IEEE, 2023.

[Oriol Vinyals et al., 2019] Oriol Vinyals et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019.

[Raffin *et al.*, 2021] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[Ribeiro *et al.*, 2016] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *SIGKDD*, pages 1135–1144. ACM, 2016.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

[Silver *et al.*, 2018] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[Tappler *et al.*, 2022] Martin Tappler, Filip Cano Córdoba, Bernhard K. Aichernig, and Bettina Könighofer. Search-based testing of reinforcement learning. In *IJCAI*, pages 503–510. ijcai.org, 2022.

[Tappler *et al.*, 2024a] Martin Tappler, Edi Muskardin, Bernhard K. Aichernig, and Bettina Könighofer. Learning environment models with continuous stochastic dynamics - with an application to deep RL testing. In *IEEE Conference on Software Testing, Verification and Validation, ICST 2024*, pages 197–208. IEEE, 2024.

[Tappler *et al.*, 2024b] Martin Tappler, Andrea Pferscher, Bernhard K. Aichernig, and Bettina Könighofer. Learning and repair of deep reinforcement learning policies from fuzz-testing data. In *ICSE*, pages 6:1–6:13. ACM, 2024.

[Torrey and Shavlik, 2010] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global, 2010.

[Towers *et al.*, 2024] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024.

[Uehara *et al.*, 2022] Masatoshi Uehara, Chengchun Shi, and Nathan Kallus. A review of off-policy evaluation in reinforcement learning. *CoRR*, abs/2212.06355, 2022.

[van Hasselt *et al.*, 2016] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, pages 2094–2100. AAAI Press, 2016.

[Watkins and Dayan, 1992] Christopher J. C. H. Watkins and Peter Dayan. Technical note Q-learning. *Mach. Learn.*, 8:279–292, 1992.

[Welch, 1947] B. L. Welch. The generalization of 'student's' problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35, 1947.

[Xie *et al.*, 2011] Xiaoyuan Xie, Joshua Wing Kei Ho, Christian Murphy, Gail E. Kaiser, Baowen Xu, and Tsong Yueh Chen. Testing and validating machine learning classifiers by metamorphic testing. *J. Syst. Softw.*, 84(4):544–558, 2011.

[Zambaldi *et al.*, 2018] Vinícius Flores Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David P. Reichert, Timothy P. Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew M. Botvinick, Oriol Vinyals, and Peter W. Battaglia. Relational deep reinforcement learning. *CoRR*, abs/1806.01830, 2018.

[Zhang *et al.*, 2022] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Trans. Software Eng.*, 48(2):1–36, 2022.

[Zolfagharian *et al.*, 2023] Amirhossein Zolfagharian, Manel Abdellatif, Lionel C. Briand, Mojtaba Bagherzadeh, and Ramesh S. A search-based testing approach for deep reinforcement learning agents. *IEEE Trans. Software Eng.*, 49(7):3715–3735, 2023.