Certification-Guided Evaluation of Reinforcement Learning Generalization

Vignesh Subramanian[1], **Djordje Zikelic**[2], **Suguman Bansal**[1]

[1]School of Computer Science, Georgia Institute of Technology, USA
[2]School of Computing and Information Systems, Singapore Management University, Singapore
[1]{vignesh, suguman}@gatech.edu, [2]dzikelic@smu.edu.sg

Abstract

This work presents a logic-driven framework to evaluate the performance of reinforcement learning (RL) algorithms in terms of their ability to generalize to unseen tasks. Our framework defines a family of inductive reachability tasks, characterized by structural similarities in task dynamics, enabling evaluation of generalization capabilities. We introduce a neural certificate function that validates trajectories generated by RL algorithms by enforcing key conditions, thereby serving as a litmus test for RL generalization. We empirically demonstrate our method's capability in certifying generalization for both generalizable RL algorithms, such as GenRL, VariBAD, PSMP and C-Learning and standard RL baselines, such as PPO and ARS across diverse benchmarks, including Car-Parking, Reacher, and Fetch Reach environments. Results show that a lower percentage of certificate function violations correlates with a higher number of test tasks successfully solved, highlighting the effectiveness of our framework in evaluating and distinguishing generalization capabilities of RL algorithms. This work provides a principled approach for benchmarking RL generalization.

1 Introduction

The utilization of neural network function approximation has significantly boosted the performance of reinforcement learning (RL) algorithms. Deep RL has achieved impressive results on large or continuous state-space tasks, such as playing games (Silver et al. 2017), autonomous system control (Levine et al. 2016), or robotics tasks (Mnih et al. 2015). However, the utilization of deep neural networks also introduces its own challenges, such as sensitivity to adversarial perturbations in the environment or state observations (Huang et al. 2017; Lechner et al. 2023), or estimation bias due to limited exploration (van Hasselt, Guez, and Silver 2016). These factors negatively impact the ability of deep RL policies to generalize to unseen environments and tasks. *Generalization* is a highly desirable property of RL agents, allowing them to adapt to and perform well in environments and tasks that are semantically similar yet not identical to those seen during the training phase (Malik, Li, and Ravikumar 2021).

Challenge – Evaluation of RL Generalization. The importance of learning RL policies with good generalization properties is widely recognized within the RL community (Malik, Li, and Ravikumar 2021; Kirk et al. 2023a; Korkmaz 2024). The problem of training RL agents with good generalization properties, which we refer to as generalizable RL agents, has been studied under several settings, such as meta RL (Beck et al. 2023) or zero-shot generalization (Kirk et al. 2023b). While these works present significant advances in *training* generalizable RL agents, the problem of *evaluating* the ability of RL agents to generalize to unseen environments and tasks has received little attention. To the best of our knowledge, no principled approach to the problem of *evaluating and comparing generalization properties of different RL agents* has been proposed. Currently, the best that one can do is to simply test different RL agents on a large number of unseen environments, and to check which RL agent exhibits the best generalization properties on the tested environments. However, the lack of a more principled comparison leads to two significant challenges. First, we need to perform testing on a *large number of new environments* in order to be able to make an informed decision on which RL agent is best at generalizing to unseen environments and tasks. Second, for RL agents that do not exhibit good generalization properties, current methods provide no means of identifying behaviours that lead to bad generalization. Ideally, we would like to be able to *flag state-action pairs that lead to incorrect generalization*. This information can then be used to relearn and ultimately repair an RL agent. Our goal is to address these challenges and to propose a principled framework for comparing generalization properties of different RL agents.

Our Contributions. In this work, we propose a novel framework for evaluating and comparing generalization properties of RL agents with respect to inductive reachability tasks. *Inductive reachability tasks* are a family of tasks defined over the same Markov decision process (MDP), where each task is specified by an initial state distribution and a sequence of goal regions that need to be reached in the given order. These tasks represent a natural target for studying RL generalizability due to the fact that, intuitively, optimal policies for inductive reachability tasks with similar initial state distributions and goal regions should also be similar.

We consider the setting in which we are given a finite set of RL agents $\pi_1, \ldots, \pi_n$ and a family Φ of inductive reachability tasks over an MDP. Each RL agent π_i is regarded as a *global policy*, meaning that it produces a control policy π_i^ϕ for each task $\phi \in \Phi$. The RL agents may be trained via any generalizable RL algorithm. Then, our goal is to evaluate and compare the performance of each RL agent on *unseen tasks* from the family Φ, i.e. tasks that were not utilized in the training procedures of any of the RL agents.

In order to evaluate and compare generalization properties of different RL agents, we introduce *certificates of correct generalization*. Our certificate is defined with respect to a finite set of *training tasks*, where each training task is a pair consisting of an inductive reachability task in Φ and a trajectory that solves the task. Then, the certificate of correct generalization with respect to this set of training tasks is a function $\mathcal{C}$ that assigns a real value to each MDP state and which is required to be (1) non-negative at every MDP state, and (2) for every training task trajectory, the value of the certificate $\mathcal{C}$ strictly decreases along the trajectory at all states that do not belong to the goal regions of reachability tasks (Definition 3.2). Intuitively, due to the non-negativity and the strict decrease conditions, the existence of such a certificate guarantees that all goal regions must be reached within finitely many time steps. We prove that a policy solves an inductive reachability task if and only if it admits a certificate of correct generalization for that task (Theorem 3.3).

We utilize our notion of certificates to compare generalization properties of RL agents as follows. First, a certificate is trained on a finite number of training tasks, by designing a loss function that enforces the defining certificate conditions at states along the training task trajectories. Second, the RL agents $\pi_1, \dots, \pi_n$ are evaluated and compared by running them on a number of *test tasks*, which are unseen tasks that were not utilized in the certificate training process. For each RL agent, we count the number of violations of the defining certificate conditions along trajectories produced by the agent for each test task. Finally, we conclude that agents leading to a smaller number of violations of the defining certificate conditions have better generalization properties.

We implemented and experimentally evaluated our method across a diverse set of RL environments. For each environment, we first trained a certificate by collecting a number of training task trajectories. Then, we used our trained certificate to evaluate the generalizability properties of different RL algorithms collected from the generalizable RL literature. Our results consistently show that a smaller number of certificate condition violations correlates with better generalizability properties of RL agents.

Our contributions can be summarized as follows:

1. We introduce the notion of *certificates of correct generalization* for inductive reachability tasks. We prove that a policy solves an inductive reachability task if and only if it admits a certificate of correct generalization for that task. Our certificates of correct generalization allow us to compare generalization properties of different RL agents.
2. We propose a training procedure for learning neural network certificates of correct generalization from a finite set of training tasks and their corresponding trajectories.
3. Our experimental evaluation demonstrates that a low number of certificate condition violations by an RL agent consistently correlates with the good generalization abilities on a wide range of inductive reachability tasks.

Related Work. The field of generalizable RL aims to develop agents capable of adapting to unseen tasks with minimal retraining. A key approach in this area is meta-learning, where the agent learns to generalize across a distribution of tasks by acquiring task-specific policies or representations. Works such as MAML (Finn, Abbeel, and Levine 2017) and its variant (Nagabandi, Finn, and Levine 2019) focus on gradient-based meta-learning for rapid adaptation to new tasks. These methods have inspired zero-shot RL approaches, where agents generalize to unseen tasks without additional training. For example, VariBAD (Zintgraf et al. 2021) leverages variational inference for task embeddings to achieve zero-shot generalization. Methods like PSMP (Inala et al. 2020) and GenRL (Subramanian et al. 2024) propose inductive structure for task families to enhance knowledge transferability.

Certificates (or *certificate functions*) provide a tool for reasoning about the correctness of control policies that has recently gained prominence within the AI and control theory communities (Dawson, Gao, and Fan 2023). In order to show that an agent satisfies some specification, existing methods compute a certificate function for that specification, which acts as a proof of satisfaction of the property of interest. The problem of learning neural network certificates for proving properties of neural controllers has been studied in the context of reachability, safety and stability tasks in deterministic (Chang, Roohi, and Gao 2019; Abate et al. 2021; Edwards, Peruffo, and Abate 2024; Zhang et al. 2023) and stochastic (Lechner et al. 2022; Zikelic et al. 2023; Mathiesen, Calvert, and Laurenti 2023; Chatterjee et al. 2023) environments. However, all these methods consider the problem of ensuring property satisfaction with respect to a *single control policy and a single task*. To the best of our knowledge, no prior work has considered the use of certificates for reasoning about and evaluating RL generalizability.

2 Preliminaries

Markov Decision Process (MDP). RL environments are formally modeled via *Markov decision processes*. A Markov decision process $\mathcal{M}$ is a tuple (S, A, P), where S is a set of states, A is a set of actions, and $P : S \times A \times S \to \mathbb{R}_{\geq 0}$ is the probabilistic transition function. We use $P(\cdot \mid s, a)$ to denote the probability distribution of the successor state after taking action a in state s. In this work, we restrict our attention to *deterministic MDPs*, meaning that each $P(\cdot \mid s, a)$ is a Dirac distribution assigning probability mass 1 to a single state s' and probability mass 0 to every other state. By a slight abuse of notation, we write $s' = P(s, a)$.

A *trajectory* ζ in an MDP is either an infinite sequence $(s_t, a_t)_{t=0}^{\infty}$ or a finite sequence $(s_t, a_t)_{t=0}^{T}$ of state-action pairs. A (pure positional) *policy* in an MDP is a map $\pi : S \to A$, which to each state assigns an action to be taken. For simplicity, we denote the trajectory obtained by a pure positional policy π by $(s_t)_{t=0}^{\infty}$ where $s_{j+1} = P(s_j, \pi(s_j))$.

Reachability Tasks. This work focuses on reachability tasks. Given an MDP with states S, a reachability task is given by the tuple (G, η) comprising of goal states $G \subseteq S$ and initial state distribution η over states S. The objective of the reachability task is to reach a state in the goal set G by starting from a state sampled from the initial state distribution η. Formally, a trajectory $\zeta = (s_t, a_t)_{t=0}^{\infty}$ satisfies a reachability task (G, η), denoted $\zeta \models (G, \eta)$, if $s_0 \sim \eta$ and there exists $t \in \mathbb{N}$ such that $s_t \in G$.

3 Certifying Inductive Tasks

This section develops a litmus test for generalizable reinforcement learning. Our notion of generalization is based on similarities between tasks. For this, we utilize the notion of inductive tasks (Subramanian et al. 2024), which are a family of tasks that are structurally similar but differ inductively in the low-level details. Specifically, we consider *inductive reachability tasks*, where each task is specified by an initial state distribution and a sequence of goal regions that need to be reached in the given order. We are guided by the intuition that these tasks are so similar that their policies should also be similar. These similarities in tasks and their policies lend inductive reachability tasks to be a good fit to evaluate the generalizability of RL algorithms.

3.1 Inductive Reachability Tasks

Inductive reachability tasks are a family of tasks over the same MDP, where each task is a sequence of reachability tasks such that the subsequent reachability task builds upon the previous one by progressively updating either the goal conditions or the initial state distribution or both. For a set S, denote by $\mathcal{P}(S)$ the set of all subsets of S and by $\mathcal{D}(S)$ the set of all probability distributions over S.

Definition 3.1 (Inductive Reachability Task). *Consider an MDP with a set of states S. An* inductive reachability task *is given by a tuple $\mathcal{T} = (\mathcal{T}_0, \mathsf{update\_goal}, \mathsf{update\_init})$, where (1) $\mathcal{T}_0 = (G_0, \eta_0)$ is the* base task *with initial goal states $G_0 \subseteq S$ and initial state distribution η_0, (2) $\mathsf{update\_goal} : \mathcal{P}(S) \mapsto \mathcal{P}(S)$ is the* goal update function, *and (3) $\mathsf{update\_init} : \mathcal{D}(S) \mapsto \mathcal{D}(S)$ is the* initial distribution update function. *Then, the inductive task $\mathcal{T}$ denotes the sequence of reachability tasks $\mathcal{T}_0 = (G_0, \eta_0), \mathcal{T}_i = (G_1, \eta_1), \cdots$ where $\mathcal{T}_0$ is defined as above and $\mathcal{T}_{i+1} = (\mathsf{update\_goal}(G_i), \mathsf{update\_init}(\eta_i))$ for $i > 0$.*

Thus, the $(i + 1)$-st reachability task in the sequence is obtained inductively from the i-th task instance, by applying the update functions to the goal states and the initial state distribution.

Example. We present an example situation (Figure 1) to demonstrate that inductive reachability tasks naturally arise in real life scenarios. Consider a two-arm robot with the objective to displace a tower of boxes from one location to another. This task can be viewed as a sequence of reachability tasks, where the objective of the i-th task $\mathcal{T}_i$ is to displace the i-th box from the top of the source tower to the top of the target tower. Formally, the i-th task instance $\mathcal{T}_i$ is a reachability task with the initial state distribution referring to the configurations in which the top i boxes have been displaced from the source tower to the target tower and the goal states referring to the configuration where the top $i + 1$ boxes have been displaced from the source tower to the target tower. In other words, the initial state distribution and the goal states refer to the configurations immediately before and after the i-th box from the top of the source tower is displaced to the target tower.

Then, this sequence of reachability tasks can be encoded as an inductive reachability task. The base task refers to the displacement of the top most box from the source tower. The initial state distribution update function formally encodes that the source tower is decreasing in height, whereas the update goal function formally encodes that the target tower is increasing in height.

3.2 Certificates for Inductive Reachability Tasks

We now define our certificates for inductive reachability tasks based on a set of task-trajectory pairs. Let us set up some notation. Let $\mathcal{T}_0, \mathcal{T}_1, \ldots$ denote an inductive reachability task with $\mathcal{T}_i = (G_i, \eta_i)$ for all $i \geq 0$. Let $\xi = \{(\mathcal{T}_0, \zeta_0), (\mathcal{T}_1, \zeta_1), \ldots\}$ be the set of task-trajectory pairs, where $\zeta_i = (s_{i,0}, s_{i,1}, \ldots, s_{i,l_i})$ is a trajectory satisfying the task $\mathcal{T}_i$, i.e., $s_{i,l_i} \in G_i$.

Definition 3.2 (Certificates for Inductive Reachability Tasks)**.** *A certificate for an inductive reachability task is a function $\mathcal{C} : S \times \mathbb{N} \to \mathbb{R}$ such that the following conditions hold for every $(\mathcal{T}_i, \zeta_i) \in \xi$:*

1. *(Non-Negativity Condition) For all states $s \in S$ and task indices $i \in \mathbb{N}_0$, the certificate satisfies: $\mathcal{C}(s, i) > 0$.*
2. *(Strict Decrease Within a Task Instance) For each $i \in \mathbb{N}_0$, there exists $\varepsilon_i > 0$ such that for all $s_{i,j}, s_{i,j+1} \in S \setminus G_i$: $\mathcal{C}(s_{i,j}, i) > \mathcal{C}(s_{i,j+1}, i) + \varepsilon_i$.*
3. *(Decrease Across Task Instances) For each $i \in \mathbb{N}_0$, the certificate satisfies: $\mathcal{C}(s_{i,l_i}, i) > \mathcal{C}(s_{i+1,0}, i + 1)$.*

Intuitively, conditions (1) and (2) let the certificate be to indicate an overapproximation of the distance of a state in the MDP from the goal state of the i-th reachability task. Observe that condition (2) requires that the distance be reduced as the policy executes from any state. Condition (3) simply requires that the values of the certificate function decrease across task instances.

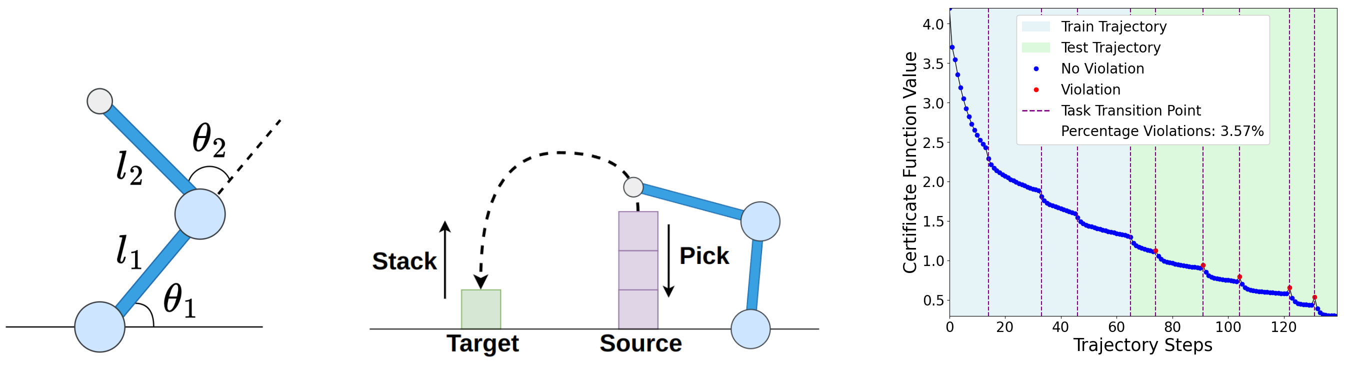
Next, we associate certificates of inductive reachability tasks with the satisfaction of the reachability task by the policies. We can prove that the existence of a certificate is a necessary and sufficient for all the policies to satisfy their respective tasks with probability 1. I.e.

Theorem 3.3. *Let $\mathcal{T}$ be an inductive reachability task with task sequence $\mathcal{T}_0, \mathcal{T}_1, \cdots$. Then a certificate for inductive reachability task exists iff*

$$\Pr[\zeta_i \models \mathcal{T}_i] = 1 \textit{ for all } i \in \mathbb{N}$$

where $\zeta_i = s_{i,0}, s_{i,1}, \cdots$ is the trajectory obtained by executing the policy π_i from an initial state sampled from η_i i.e. $s_{i,0} \sim \eta_i$.

Proof Sketch. Suppose first that a certificate for an inductive reachability task exists. Then, the non-negativity and decrease conditions of certificates together imply that each



(a) Agent dynamics *(b)* Inductive Task *(c)* Certificate function values generated by Varibad algorithm on Reacher environment

Figure 1. Tower Destacking: The task is to pick boxes from *Source* and stack it on *Target*.

goal region must be reached in finitely many steps. Hence, the inductive reachability task is satisfied. Conversely, suppose that an inductive reachability task is satisfied by a trajectory. Consider any strictly monotonically decreasing sequence $(x_i)_{i=1}^{\infty}$ of positive real numbers. Then, using the notation from Definition 3.2, a certificate $\mathcal{C}$ can be defined by letting $\mathcal{C}(s_{i,j}, i) = x_{i+1} + (x_i - x_{i+1}) \cdot (l_i - j)/l_i$ along the trajectory states and letting $\mathcal{C}(s, i) = 0$ otherwise. □

3.3 Evaluation of RL Generalizability via Certificates

We now present our framework for evaluating and comparing generalizability of different RL agents by utilizing certificates of correct generalization introduced in the previous section. Our framework consists of two phases: (1) Certificate Training, in which a neural network certificate is trained based on a finite number of given training task, and (2) Generalizability Evaluation, in which the previously trained certificate is used to evaluate generalizability properties of RL agents.

Certificate Training. We first train a neural network certificate $\mathcal{C} : S \times \mathbb{N} \to \mathbb{R}$. Our training procedure takes as input a finite number of *training task*, where each training task is a pair consisting of an inductive reachability task and a trajectory that solves the task. These training tasks are utilized to design a loss function for training our neural certificate. Intuitively, the loss function encodes the defining certificate conditions in Definition 3.2 by enforcing (strict) decrease of $\mathcal{C}$ for each pair of successor states along the trajectories. Minimizing the loss function then corresponds to minimizing the number of violations of these defining conditions across training task trajectories. The details of the training procedure are provided to Section 3.4. In practice, the trajectories for training tasks are generated either by an RL algorithm or by an oracle.

Generalizability Evaluation. Once trained, the neural certificate is utilized to evaluate the generalizability properties of given RL agents $\pi_1, \ldots, \pi_n$ as follows.

First, a finite number of test inductive reachability tasks Φ^{test} are chosen to assess generalizability of given RL agents. We require that these test tasks are not part of the training set used to learn the certificate. Second, for each RL agent π_i and for each test task $\phi \in \Phi^{\text{test}}$, we use π_i^{ϕ} to generate a trajectory $\xi_{\pi_i}^{\phi}$. We then count the number of violations of the certificate decrease conditions of $\mathcal{C}$ along $\xi_{\pi_i}^{\phi}$ (conditions 2 and 3 in Definition 3.2), and define $N_{\pi_i}^{\phi}$ to be the total number of violations. Finally, for each RL agent π_i, we define

$$N_{\pi_i} = \sum_{\phi \in \Phi^{\text{test}}} N_{\pi_i}^{\phi}$$

We then conclude that RL agents π_i with the lower number of violations N_i have better generalizability properties.

This evaluation framework provides a principled method to assess the performance of RL agents on unseen tasks, using the neural certificate as a validation tool. By comparing the percentage of certificate condition violations across different RL algorithms, we gain insights into their ability to generalize to new tasks and adhere to the structural patterns of the inductive task family.

3.4 Learning Neural Network Certificates

Training Data. The objective is to learn a certificate function $\mathcal{C}(s, i)$ that can validate trajectories generated by RL algorithms within an inductive task family. The training tasks for the certificate are a subset of the inductive task family, denoted as $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{n-1}$, where each task $\mathcal{T}_i$ involves a specific goal set G_i and an initial state distribution η_i. The training trajectories $\zeta_i = (s_{i,0}, s_{i,1}, s_{i,2}, \ldots, s_{i,l_i})$ are generated by executing an RL policy π_i trained to solve the corresponding tasks, and these trajectories satisfy the reachability conditions for their respective goals. Once trained on these trajectories, the certificate function $\mathcal{C}(s, i)$ is used to validate unseen test trajectories generated by RL algorithms for tasks $\mathcal{T}_n, \mathcal{T}_{n+1}, \ldots$ from the same inductive task family.

Training Procedure. We learn the certificate function $\mathcal{C}(s, i)$, which minimizes a loss function designed to enforce a monotonically decreasing property across trajectories and within trajectories. Specifically, the certificate $\mathcal{C}(s, i)$ should produce a sequence of values that decreases as it progresses along a trajectory, satisfying the decreasing condition both within individual trajectories and across tasks. The loss function for this objective is defined as:

$$L = \sum_{i=0}^{n} \left(\underbrace{\sum_{j=0}^{l_i} \max\{0,\, \mathcal{C}(s_{i,j+1}, i) + \varepsilon_i - \mathcal{C}(s_{i,j}, i)\}}_{\text{Penalizes to Enforce (2)}} \right.$$

$$\left. + \underbrace{\sum_{\substack{s \in G_i \\ t \sim \eta_{i+1}}} \max\{0,\, \mathcal{C}(t, i+1) - \mathcal{C}(s, i)\}}_{\text{Penalizes to Enforce (3)}} \right).$$

Here, n is the total number of tasks in the inductive task family, l_i is the length of the trajectory ζ_i for the task $\mathcal{T}_i$, and $\mathcal{C}(s, i)$ is the certificate value for state s in the context of the i-th task. The first summation enforces the decrease condition (2) within each task instance by penalizing any increases in $\mathcal{C}$ along trajectories. The second summation enforces the across-task decrease condition (3) by ensuring the values assigned to goal states of $\mathcal{T}_i$ exceed those of the next task's start distribution. Minimizing this loss guides the model toward a function $\mathcal{C}(s, i)$ that satisfies both the within-task and across-task monotonicity conditions described in the certificate definition.

To achieve this, the certificate $\mathcal{C}(s, i)$ is modeled using a *Long Short-Term Memory* (Hochreiter 1997) (LSTM) network, which is well-suited for processing sequential data like trajectories. The LSTM acts as the function $\mathcal{C}(s, i)$, taking as input the state s and the task index i, and outputting a real number representing the certificate value. The LSTM is trained on a concatenated sequence of trajectories $(\zeta_0, \zeta_1, \ldots, \zeta_n)$, where $\zeta_i = (s_{i,0}, s_{i,1}, \ldots, s_{i,l_i})$ represents the trajectory for task $\mathcal{T}_i$ generated by the policy π_i. Each state s_{ij} is concatenated with the corresponding task index i and passed as input to the LSTM, which outputs $\mathcal{C}(s_{i,j}, i)$, the certificate value for that state. The concatenated trajectories ensure that the LSTM is trained on the inductive task family, enabling it to model the dynamics of trajectories across tasks.

The intuition behind using the LSTM architecture to model certificates $\mathcal{C}(s, i)$ is motivated by its ability to capture sequential dependencies in data. Trajectories represent ordered sequences of states, where the progression along the trajectory encodes information about task dynamics and the proximity to the target state. LSTMs are well-suited to learn these patterns due to their inherent design, which maintains memory of previous states and captures long-term dependencies. During training, the LSTM learns to output monotonically decreasing certificate values along each trajectory, ensuring the decreasing condition is satisfied. Once trained, the LSTM is applied to unseen test trajectories from tasks $\mathcal{T}_N$ and beyond, which were not included in the training set. On these unseen test trajectories, the LSTM evaluates whether the decreasing condition holds. If the certificate values decrease consistently along the unseen test trajectory, it confirms the trajectory's validity with respect to reachability. Conversely, violations of the decreasing condition indicate deviations or poor-quality trajectories, thus failing the reachability criteria. By leveraging the LSTM's sequential modeling capabilities, this approach ensures that the certificate $\mathcal{C}(s, i)$ generalizes effectively to unseen tasks, providing a principled mechanism for validating trajectories across the task family.

4 Experiments

4.1 Experimental Setup

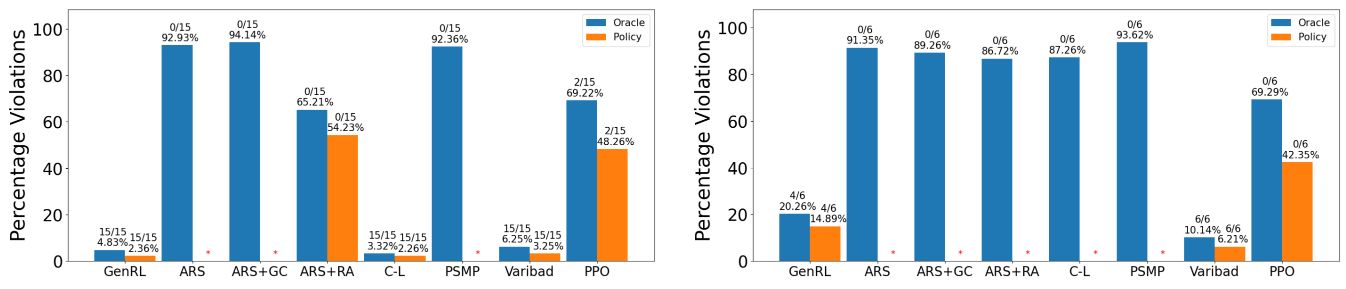
The goal of our experiments is to address the following two research questions:

- **RQ1:** Does minimizing the number of certificate violations correspond to better generalization properties of RL agents? In other words, can our certificate-based method serve as an effective litmus test for evaluating the generalizability of RL algorithms?
- **RQ2:** Is the training procedure described in Section 3.4 able to learn effective certificates in practice, capable of validating trajectories with minimal violations?

To investigate these questions, we focus on learning a certificate function $\mathcal{C}(s, i)$ that validates the correctness of trajectories generated by RL algorithms within inductive reachability task families. Specifically, the experiments assess the generalization of RL agents to unseen test tasks by evaluating the number of violations of the certificate's decreasing condition across trajectories. A lower number of violations indicates better alignment with the reachability criteria, suggesting stronger generalization capabilities of the RL policies.

To evaluate this, we divide the tasks into two distinct sets: **training tasks** and **unseen test tasks**. The training tasks, denoted as $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{n-1}$, are a subset of the inductive reachability task family and are used to generate trajectories that the certificate is trained on. The certificate is modeled using a LSTM network, which processes sequential data from these training trajectories and learns patterns that satisfy the reachability criteria. The unseen test tasks, denoted as $\mathcal{T}_n, \mathcal{T}_{n+1}, \ldots$, belong to the same inductive reachability task family but are not part of the training set. The LSTM-trained certificate is then applied to validate the correctness of unseen test trajectories by checking whether they satisfy the decreasing condition.

We consider two options for generating training trajectories to learn the certificate: (1) *RL-generated trajectories*, where the trajectories are obtained from policies traiend using an RL algorithm or (2) *oracle-generated trajectories*, which are trajectories obtained from a planning algorithm (i.e. the dynamics of the environment is known). Oracle-generated trajectories serve as ideal ground-truth solutions that perfectly satisfy reachability criteria. The choice between RL-generated and oracle-generated trajectories is exclusive, meaning only one type is used for training in a given setup. While RL-generated trajectories align the certificate with the behavior of specific RL algorithms, oracle trajectories provide a strict baseline to compare RL generalization against ideal behavior.



(a) Car-Parking Moving Initial Point

(b) Reacher Two Arm Robot

Figure 2

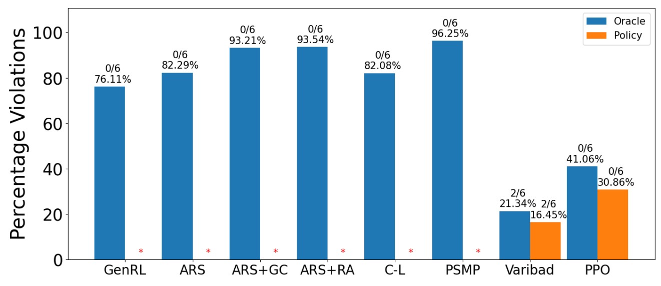


Figure 3. Fetch Reach

Evaluation Environments. We evaluate our approach across diverse environments: **Car-Parking**, **Reacher Two-Arm Robot**, and **Fetch Reach**. Each environment features a unique set of dynamics and state spaces:

- **Car-Parking** (Inala et al. 2020): A 2D navigation and parking task where a car, modeled as a rectangular entity with orientation, is moved from an initial position to a goal position.
- **Reacher Two-Arm Robot** (Fig. 1): A planar robot with two arms tasked with transferring boxes from a source stack to a target stack. The tasks vary by adjusting the initial and target positions, challenging the robot to adapt its movements to new configurations.
- **Fetch Reach** (Brockman et al. 2016): A 7-DOF robotic arm tasked with moving its end effector from an initial position to a goal position in a continuous 3D space.

Further details about the environments are provided in Appendix 1.

RL Algorithms. We evaluate several RL algorithms to assess the effectiveness of the certificate function $\mathcal{C}(s, i)$ in validating trajectories across both training and unseen test tasks. These include **generalizable RL algorithms** such as GenRL (Subramanian et al. 2024) (inductive generalization), Varibad (Zintgraf et al. 2021) (meta-learning), PSMP (Inala et al. 2020) (inductive generalization), and C-Learning (Naderian et al. 2021) (goal-conditioned generalization). These algorithms are designed to adapt to new tasks with minimal retraining, making them particularly relevant for evaluating generalization in inductive task families. In addition to generalizable algorithms, we evaluate **standard RL algorithms** such as Augmented Random Search (ARS) (as well as its variants: ARS + Reward Aggregations, ARS + Goal Conditioning) and Proximal Policy Optimization (PPO) (Schulman et al. 2017). ARS and PPO are widely used baselines in RL, known for their strong performance on a variety of tasks but without explicit mechanisms for generalization. By including these algorithms, we aim to contrast the behavior of generalizable RL methods with standard RL techniques, particularly in their ability to generate trajectories that adhere to the reachability criteria as validated by the certificate.

Evaluation Protocol. To evaluate the generalization capabilities of RL algorithms and validate the effectiveness of the certificate function $\mathcal{C}(s, i)$, we conduct experiments using inductive reachability tasks. For the **Car-Parking** environment, we use 5 training tasks and 15 unseen test tasks. In the **Reacher** and **Fetch-Reach** environments, we use 4 training tasks and 6 unseen test tasks each. The training tasks are sampled from the inductive reachability task family and are used to train the certificate function $\mathcal{C}(s, i)$ using trajectories generated by the RL algorithms. The unseen test tasks are distinct from the training set and are used to evaluate the generalization of RL agents to new tasks within the same inductive task family.

Metrics. For each RL algorithm, we assess its performance on unseen test tasks using two key metrics:

1. **Number of Successfully Solved Test Tasks:** A test task is considered solved if the trajectory generated by the RL algorithm reaches the goal region without violating task constraints. This metric provides a direct measure of the RL algorithm's ability to adapt to unseen tasks.
2. **Percentage of Certificate Violations:** This metric quantifies the percentage of successor state pairs along test trajectories that violate the certificate's decreasing condition. A lower percentage indicates better alignment with the reachability criteria encoded in the certificate, suggesting higher generalization capability.

To address **RQ1**, we analyze the correlation between these two metrics. A higher count of successfully solved test tasks accompanied by a lower percentage of violations demonstrates the certificate's effectiveness as a litmus test for RL

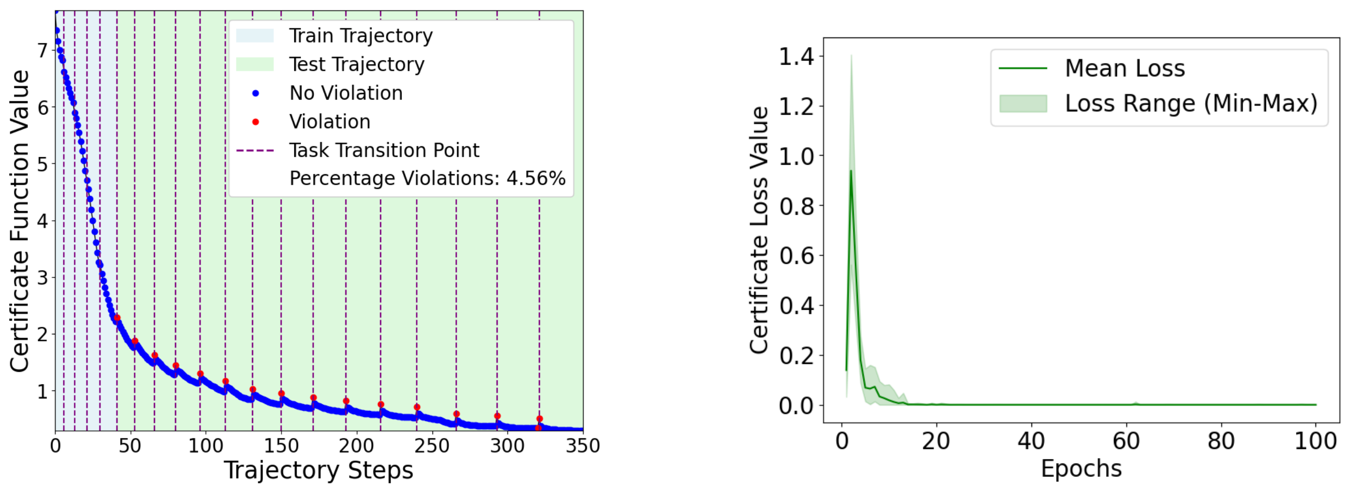


Figure 4. Car-Parking Environment: Certificate Function Values and Loss Function Values

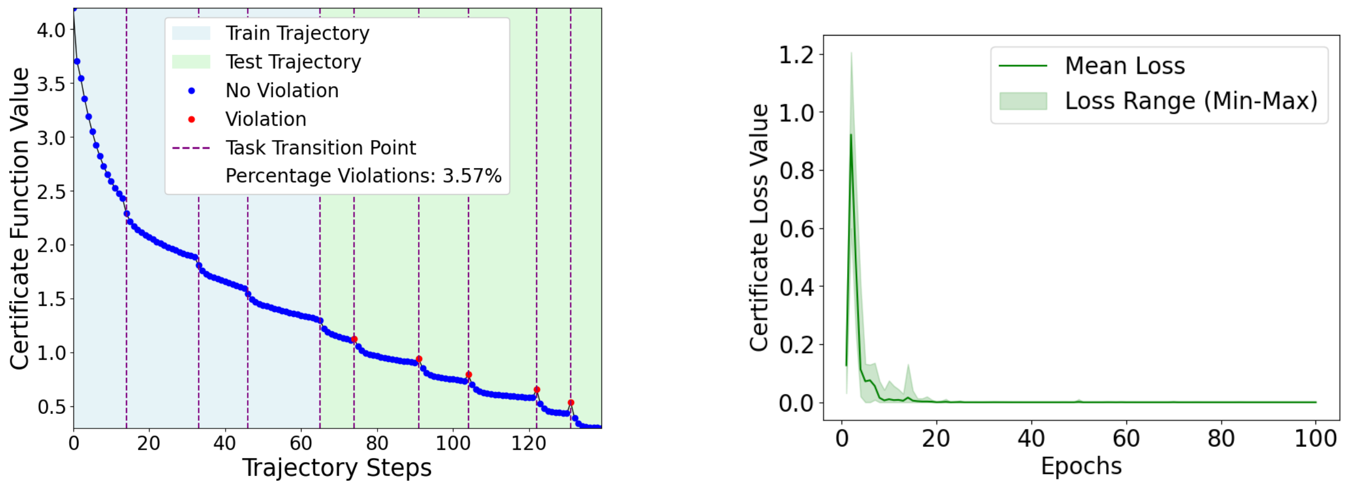


Figure 5. Reacher Environment: Certificate Function Values and Loss Function Values

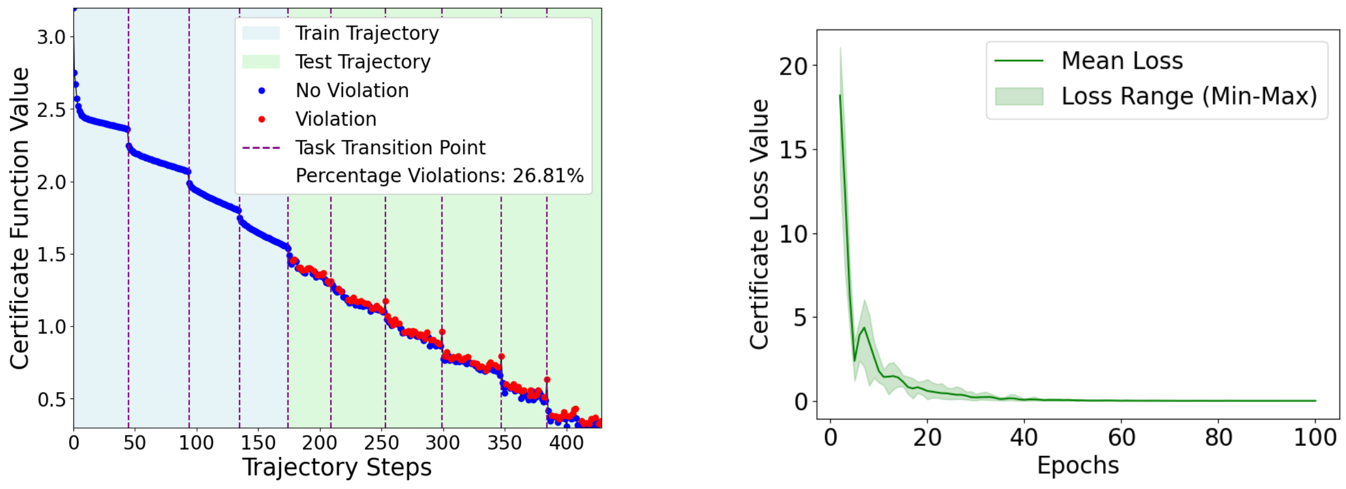


Figure 6. Fetch-Reach Environment: Certificate Function Values and Loss Function Values

generalization. To further ensure robustness, each experiment is repeated 10 times, and the mean performance across these runs is reported to illustrate the results.

4.2 Results

Comparison of the generalizability of the algorithms using certificates. Figures 2a, 2b, and 3 present the percentage of violations in the decreasing condition of the certificate when validating test trajectories in the car parking, reacher robot and fetch-reach environments. The bar graph shows two categories for each algorithm: results when test trajectories are evaluated using certificates trained on RL-generated training trajectories (orange) and certificates trained on oracle-generated training trajectories (blue). Each bar is labeled with two metrics: the number of test tasks that passed the validation (shown as a fraction) and the percentage of violations. ”Passed” means that the test tasks exhibited a sufficiently low percentage of violations, indicating that the trajectories generated by the RL algorithm aligned well with the reachability criteria enforced by the certificate. For instance, a bar labeled $0/15$ and 92.93% indicates that no test tasks passed validation, and 92.93% of the certificate's decreasing condition checks were violated. In cases where a star (*) appears on the bar, it indicates that the RL algorithm did not train successfully to solve all the training inductive tasks. As a result, the certificate could not be effectively trained, leading to inherent failure in validating test tasks. This reflects the inability of the RL algorithm to generalize during training, which directly impacts the validation results.

In turn, a bar labeled $15/15$ and 4.83% shows that all the test tasks passed validation and a lower percentage of violations indicates that the RL algorithm generalizes well to solve more unseen tasks that were not part of the training distribution. This shows the effectiveness of the RL policy in adapting to new tasks within the inductive task family. As a result, RL algorithms that produce trajectories with fewer violations demonstrate stronger generalization capabilities and better alignment with the patterns encoded in the certificate.

When comparing oracle-trained certificates (blue bars) to RL-trained certificates (orange bars), we observe that oracle-trained certificates typically result in higher violations when tested against RL-generated trajectories. This is expected, as oracle trajectories represent ideal behavior, and deviations in RL-generated trajectories from the ground truth lead to more violations of the decreasing condition. In contrast, RL-generated certificates demonstrate lower violations when tested on trajectories produced by the same RL algorithm, as the certificate is trained on data that reflects the specific patterns of these trajectories followed by their RL algorithms.

Certificate Function and the Loss Evaluation. The graphs in Figures 4, 5, and 6 illustrate two key aspects of our experimental evaluation: (a) the progression of certificate function values across train and test trajectories, and (b) the loss function trends during training, which serve as a sanity check for the effectiveness of the certificate training. The left subfigure shows the progression of certificate function values $\mathcal{C}(s, i)$ over trajectory steps for train and test tasks. The blue dots represent the certificate values for the trajectories. The blue background indicates the region corresponding to training trajectories while the green background indicates the region corresponding to unseen test trajectories. Red points signify violations of the decreasing condition, where the certificate function fails to decrease monotonically. Purple dashed lines denote task transition points, where the system transitions from one task to the next within the inductive task family. The percentage of violations across all trajectory steps is also reported, providing a quantitative measure of the RL algorithm's generalizability. The right subfigure shows the loss function value during training, across epochs. The shaded region indicates the range (min-max) of the loss values, and the solid line represents the mean loss. A consistent decrease in the loss over epochs indicates successful training of the certificate.

In Figure 4, the GenRL algorithm is evaluated on the Car-Parking environment, where the certificate function values decrease strictly along the trajectory with a small percentage of violations (4.56%). The corresponding loss graph shows a rapid decrease in the loss during the initial epochs, confirming successful training of the certificate function. Similarly, in Figure 5, the Varibad algorithm is evaluated on the Reacher environment. The certificate function values decrease smoothly across both training and test trajectories, with minimal violations (3.57%), indicating strong generalization of the RL algorithm to unseen test tasks. The loss function graph for the Reacher environment also shows a rapid decrease and stabilization, further validating the certificate's training process. In Figure 6, the certificate function graph shows more frequent violations (26.81%), suggesting that the RL algorithm struggled to generalize across the unseen test tasks. The loss function's convergence shows that the certificate was trained well, hence confirming the result of the certificate function.

5 Conclusion

This work presents a novel framework for evaluating and comparing the generalization capabilities of RL agents using certificate functions. By leveraging these certificates, we provide a method to validate trajectories and assess RL agents' performance on unseen tasks. Our results demonstrate that minimizing certificate violations strongly correlates with better generalization, making this approach a reliable litmus test for evaluating RL generalizability. The experiments also confirm the effectiveness of the proposed training procedure in learning certificates that capture the underlying structure of inductive task trajectories.

References

Abate, A.; Ahmed, D.; Giacobbe, M.; and Peruffo, A. 2021. Formal Synthesis of Lyapunov Neural Networks. *IEEE Control. Syst. Lett.*, 5(3): 773–778.

Beck, J.; Vuorio, R.; Liu, E. Z.; Xiong, Z.; Zintgraf, L.; Finn, C.; and Whiteson, S. 2023. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. arXiv. *arXiv preprint arXiv:1606.01540*, 10.

Chang, Y.; Roohi, N.; and Gao, S. 2019. Neural Lyapunov Control. In Wallach, H. M.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. B.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 3240–3249.

Chatterjee, K.; Henzinger, T. A.; Lechner, M.; and Zikelic, D. 2023. A Learner-Verifier Framework for Neural Network Controllers and Certificates of Stochastic Systems. In Sankaranarayanan, S.; and Sharygina, N., eds., *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I*, volume 13993 of *Lecture Notes in Computer Science*, 3–25. Springer.

Dawson, C.; Gao, S.; and Fan, C. 2023. Safe Control With Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction Methods for Robotics and Control. *IEEE Trans. Robotics*, 39(3): 1749–1767.

Edwards, A.; Peruffo, A.; and Abate, A. 2024. Fossil 2.0: Formal Certificate Synthesis for the Verification and Control of Dynamical Models. In Ábrahám, E.; and Jr., M. M., eds., *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2024, Hong Kong SAR, China, May 14-16, 2024*, 26:1–26:10. ACM.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, 1126–1135. PMLR.

Hochreiter, S. 1997. Long Short-term Memory. *Neural Computation MIT-Press*.

Huang, S. H.; Papernot, N.; Goodfellow, I. J.; Duan, Y.; and Abbeel, P. 2017. Adversarial Attacks on Neural Network Policies. In *ICLR (Workshop)*. OpenReview.net.

Inala, J. P.; Bastani, O.; Tavares, Z.; and Solar-Lezama, A. 2020. Synthesizing programmatic policies that inductively generalize. In *8th International Conference on Learning Representations*.

Kirk, R.; Zhang, A.; Grefenstette, E.; and Rocktäschel, T. 2023a. A Survey of Zero-shot Generalisation in Deep Reinforcement Learning. *J. Artif. Intell. Res.*, 76: 201–264.

Kirk, R.; Zhang, A.; Grefenstette, E.; and Rocktäschel, T. 2023b. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76: 201–264.

Korkmaz, E. 2024. A Survey Analyzing Generalization in Deep Reinforcement Learning. *CoRR*, abs/2401.02349.

Lechner, M.; Amini, A.; Rus, D.; and Henzinger, T. A. 2023. Revisiting the Adversarial Robustness-Accuracy Tradeoff in Robot Learning. *IEEE Robotics Autom. Lett.*, 8(3): 1595–1602.

Lechner, M.; Zikelic, D.; Chatterjee, K.; and Henzinger, T. A. 2022. Stability Verification in Stochastic Control Systems via Neural Network Supermartingales. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, 7326–7336. AAAI Press.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-End Training of Deep Visuomotor Policies. *J. Mach. Learn. Res.*, 17: 39:1–39:40.

Malik, D.; Li, Y.; and Ravikumar, P. 2021. When Is Generalizable Reinforcement Learning Tractable? In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 8032–8045.

Mathiesen, F. B.; Calvert, S. C.; and Laurenti, L. 2023. Safety Certification for Stochastic Systems via Neural Barrier Functions. *IEEE Control. Syst. Lett.*, 7: 973–978.

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.

Naderian, P.; Loaiza-Ganem, G.; Braviner, H. J.; Caterini, A. L.; Cresswell, J. C.; Li, T.; and Garg, A. 2021. C-learning: Horizon-aware cumulative accessibility estimation. *International Conference on Learning Representations*.

Nagabandi, A.; Finn, C.; and Levine, S. 2019. Deep online learning via meta-learning: Continual adaptation for model-based rl. *International Conference on Learning Representations*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T. P.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of Go without human knowledge. *Nat.*, 550(7676): 354–359.

Subramanian, V.; Kushwah, R.; Roy, S.; and Bansal, S. 2024. Inductive Generalization in Reinforcement Learning from Specifications. *CoRR*, abs/2406.03651.

van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-Learning. In *AAAI*, 2094–2100. AAAI Press.

Zhang, H.; Wu, J.; Vorobeychik, Y.; and Clark, A. 2023. Exact Verification of ReLU Neural Control Barrier Functions. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zikelic, D.; Lechner, M.; Henzinger, T. A.; and Chatterjee, K. 2023. Learning Control Policies for Stochastic Systems with

Reach-Avoid Guarantees. In *AAAI*, 11926–11935. AAAI Press.

Zintgraf, L.; Schulze, S.; Lu, C.; Feng, L.; Igl, M.; Shiarlis, K.; Gal, Y.; Hofmann, K.; and Whiteson, S. 2021. Varibad: Variational bayes-adaptive deep rl via meta-learning. *Journal of Machine Learning Research*, 22(289): 1–39.

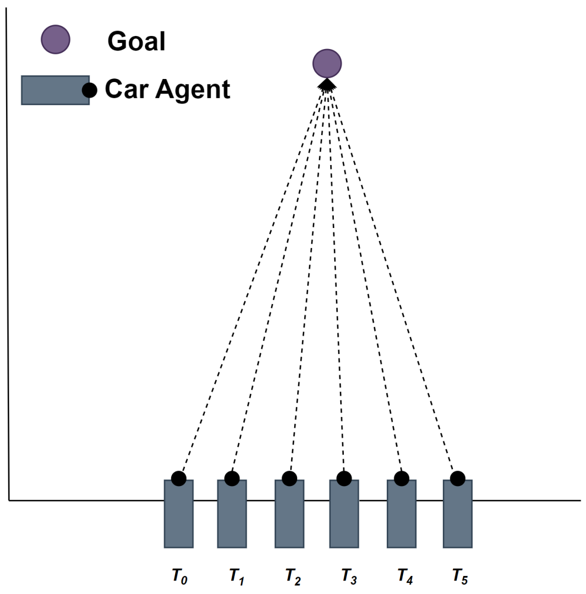
Appendix

1 Experimental Setup

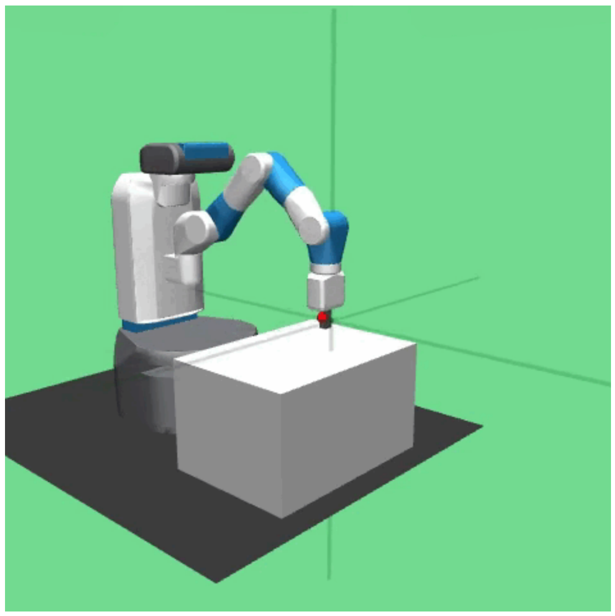
Our approach is evaluated across several environments, including the Car-Parking, Reacher Two-Arm Robot, and Fetch Reach settings, which feature diverse tasks and specifications. All tasks involve moving from an initial state to a target state where reachability is the main objective, with the environment subject to different dynamics and state spaces. The **Car-Parking environment** has a 4-dimensional state space, including x and y positions, orientation, and velocity, with 2 actions: velocity and steering direction. The **OpenAI Fetch-Reach environment** involves a robotic arm with a 8-dimensional state space, including the x, y, z positions of the end effector, state of the left and right gripper, and the x, y , z velocities of the end effector, with 3 actions: displacement along the x, y, and z axes. The **Reacher Two-Arm Robot environment** uses a 2-dimensional state space representing the x and y coordinates of the end effector, with 2 actions controlling the angles of the two joints.

The inductive tasks in these environments are systematically defined by updating the initial distributions and target goals according to the functions update_init and update_goal, respectively:

- **Car-Parking Environment** (Fig. 7a): In this environment, the initial distribution is updated by shifting the initial position along the x-axis by a fixed value C, i.e.,
$$\eta_{i+1}(s) = \eta_i(s + (C, 0)).$$
$$G_{i+1} = G_i + (0, 0).$$
- **Reacher two arm robot environment** (Fig. 1): The goal is to move a box from height $H - C$ in the *Source* stack to height $C + 1$ in the *Target* stack where H is the initial height of the source tower.
$$\eta_{i+1}(s) = \eta_i(s + (0, H - C)).$$
$$G_{i+1} = G_i + (0, C + 1).$$
- **Fetch Reach Environment** (Fig. 7b): In this environment, the inductive tasks are defined by moving the target position along the y-axis by a fixed increment C, i.e.,
$$\eta_{i+1}(s) = \eta_i(s + (0, 0, 0)).$$
$$G_{i+1} = G_i + (0, C, 0),$$
while keeping the initial distribution η_i fixed.



(a) Illustration of the CarParking Task.



(b) Illustration of the Fetch-Reach Task.

Figure 7. Illustration of the environments