Certificate-Guided Evaluation of Reinforcement Learning Generalization

Vignesh Subramanian[1] , **Đorđe Žikelić**[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
{vignesh, suguman}@gatech.edu, dzikelic@smu.edu.sg

Abstract

This work presents a logic-driven framework to evaluate the performance of reinforcement learning (RL) algorithms in their ability to generalize to unseen tasks. Our framework defines a family of inductive reach-avoid 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 several state-of-the-art generalizable RL algorithms on challenging continuous environments. Our 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

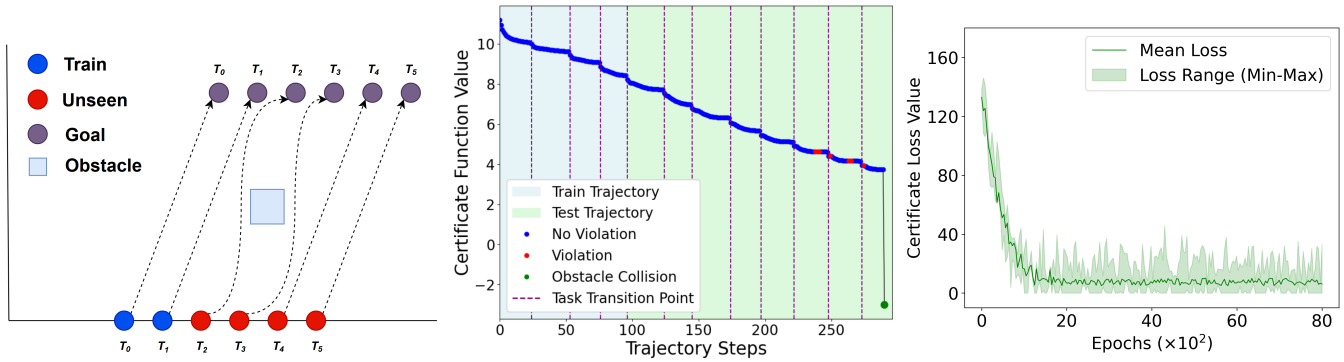
The *generalization* problem in reinforcement learning (RL) addresses how agents can learn policies that effectively transfer to new, unseen situations beyond their training experience. Successful generalization enables agents to maintain performance across environmental variations and unfamiliar states without requiring explicit training for every scenario. The importance of learning RL policies with good generalization properties is widely recognized within the RL community [Malik *et al.*, 2021; Kirk *et al.*, 2023; Korkmaz, 2024]. Various approaches of zero-shot generalization, i.e., generalization without any retraining have been studied in the literature [Beck *et al.*, 2023; Kirk *et al.*, 2023]. 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 test different RL agents on a large number of unseen environments and tasks. 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 exhibits best generalization properties. 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*, hence enhancing explainability of where RL generealization failed. Our goal is to address these challenges and to propose a principled framework for comparing generalization properties of RL agents.

In this work, we propose a novel framework for evaluating and comparing generalization properties of RL agents. Our framework considers generalization with respect to inductive reach-avoid tasks [Subramanian *et al.*, 2024]. An *inductive reach-avoid task* is a sequence of reach-avoid tasks that are structurally similar but differ in low-level details. The goal of each reach-avoid task in the sequence is to reach some goal region while avoiding some unsafe region, with subsequent reach-avoid task building upon the previous tasks inductively. Figure 1a illustrates an inductive reach-avoid task where each dotted line represents the reach-avoid task to navigate from an initial location (the lower location) to a goal location (the upper location) while avoiding the obstacle. Each task is related to the task on its right as both the initial and the goal location move to the left by one unit.

Inductive tasks are suitable to evaluate generalizability as the structural similarity of reach-avoid tasks in this inductive sequence intuitively means that their policies should also be similar. In particular, if a good generalizable RL agent is trained on few of the reach-avoid tasks in an inductive task, then the RL agent should be able to capture the *essence of the task* to generalize to the unseen tasks in the inductive family. The challenge lies in formalizing the essence of the task.

To this end, we introduce *certificates of correct generalization*. A certificate of correct generalization is defined with respect to a set of task-trajectory pairs, which serve as demonstrations of what a trajectory that satisfies a reach-avoid task should look like. Formally, it is a function $\mathcal{C}$ which to each MDP state and reach-avoid task assigns a real value, which is required to be (1) non-negative at all safe states along the demonstration trajectories, (2) strictly decreasing along each



(a) Inductive task *(b)* Certificate function values *(c)* Loss graph for learning the certificate

Figure 1. Car - Moving Initial Point and Goal Point with obstacle: The task is navigate the car from *Initial* to its respective *Goal*.

trajectory until they reach a goal state, (3) decreasing across task instances, and (4) negative at all unsafe states. These conditions together allow the certificates to capture *monotonic invariant behaviour* that RL agents with good generalizability properties should satisfy, in order to generalize well to new reach-avoid tasks. In particular, the certificate value provides a quantitative measure of *progress* along a trajectory towards satisfying its reach-avoid task.

Our framework for evaluating generalization properties of RL agents assumes that it is provided with a set of task-trajectory demonstrations, and uses them to compute a certificates of correct generalization. It then runs each RL agent on a set of new and unseen tasks from the inductive reach-avoid task family, and uses the certificate to quantify and compare the generalization capabilities of each RL agent. In particular, RL agents that lead to a lower percentage of certificate function violation (i.e. a smaller number of states along their trajectories that violate conditions (1)-(4) above) are concluded to have better generalization properties.

Finally, in order to compute our certificates, we design a *training procedure* for learning neural network certificates of correct generalization from given demonstration task trajectories, where a loss function is designed to enforce certificate conditions (1)-(4) above at states along the demonstration task trajectories. The certificate function for the inductive task in Figure 1a is shown in Figure 1b.

We implemented and experimentally evaluated our method across a diverse set of RL environments. For each environment, we first trained a neural network certificate from a number of demonstration task trajectories. Then, we used our trained certificate to evaluate 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 algorithms.

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 *et al.*, 2017] and its variant [Nagabandi *et al.*, 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. VariBAD [Zintgraf *et al.*, 2021] leverages variational inference for task embeddings to achieve zero-shot generalization. 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 *et al.*, 2023]. In order to show that an agent satisfies some specification, existing methods compute a certificate which acts as a proof that the specification is satisfies. 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 *et al.*, 2019; Abate *et al.*, 2021; Edwards *et al.*, 2024; Zhang *et al.*, 2023] and stochastic [Lechner *et al.*, 2022; Zikelic *et al.*, 2023; Mathiesen *et al.*, 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 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. RL environments are formally modeled via Markov decision processes. A *Markov decision process (MDP)* $\mathcal{M}$ 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 a 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 an infinite sequence $(s_t, a_t)_{t=0}^{\infty}$ of state-action pairs. A *finite trajectory* is a finite prefix of a trajectory. 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 $s_{j+1} = P(s_j, \pi(s_j))$.

Reach-Avoid Tasks. This work focuses on *reach-avoid* tasks. Given an MDP with states S, a reach-avoid task is defined by the tuple (G, η, X_u) comprising of *goal states* $G \subseteq S$, *initial state distribution* η over MDP states S, and *unsafe states* $X_u \subseteq S \setminus \mathrm{supp}(\eta)$ where $\mathrm{supp}(\eta)$ is the support of the initial state distribution. The objective of a reach-avoid task is to reach the goal states from the initial states while avoiding all unsafe states along the trajectory. Formally, a trajectory $\zeta = (s_t, a_t)_{t=0}^{\infty}$ satisfies a reach-avoid task (G, η, X_u), denoted $\zeta \models (G, \eta, X_u)$, if exists $T \geq 0$ such that $s_0 \sim \eta$, $s_T \in G$, and $s_j \in S \setminus X_u$ for all $0 \leq j \leq T$. For simplicity, we use the notation $X_s = S \setminus X_u$ to denote the *safe states*.

Assumption. We assume that all states in the set of unsafe states X_u are sink states. This means that once a trajectory enters any state $s \in X_u$, it cannot leave, i.e., $P(s, a) = s$ for all $s \in X_u$ and $a \in A$. This assumption reflects scenarios where entering an unsafe state represents an irreversible failure, such as collisions in robotic navigation. As a result, any trajectory that enters X_u is immediately terminated and considered invalid for the reach-avoid task.

3 Evaluation of RL Generalizability

This section presents our framework for evaluating and comparing generalization properties of RL agents. Our framework evaluates generalization of RL algorithms based on their ability to extrapolate to similar but different tasks.

To achieve this, we consider generalization with respect to *inductive tasks* ([Subramanian *et al.*, 2024]), which are a family of tasks that are structurally similar but differ inductively in the low-level details. We consider *inductive reach-avoid tasks*. We are guided by the intuition that these tasks are so similar that their policies should also be similar, making them a good fit for evaluating RL generalizability. Inductive reach-avoid tasks are formally defined in Section 3.1.

In order to evaluate generalization properties of RL algorithms, our framework assumes that it is provided with a set of task-trajectory pairs which serve as *demonstrations* of what a trajectory that satisfies some reach-avoid task looks like. It then uses these demonstrations to extract a quantitative measure of *progress* along demonstration trajectories towards satisfying their reach-avoid tasks. This progress measure is formally captured via *certificates of correct generalization*, a notion that we introduce in Section 3.2.

Finally, our framework is presented in Section 3.3. Our framework first trains a certificate of correct generalization from a given set of task-trajectory demonstration pairs. It then runs each RL agent on a set of new and unseen tasks, and uses the certificate to quantify and compare the generalization capabilities of each RL agent.

3.1 Inductive Reach-Avoid Tasks

An inductive reach-avoid task is a sequence of reach-avoid tasks over the same MDP, such that the subsequent reach-avoid task builds upon the previous one by progressively updating the goal region, the initial state distribution, or both. We assume that the set of unsafe states is the same for all tasks in the sequence, as these correspond to e.g. physical obstacles or collisions. 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 1 (Inductive reach-avoid task)**.** *Consider an MDP with a set of states S. An* inductive reach-avoid task *is given by a tuple $\mathcal{T} = (\mathcal{T}_0, \mathsf{update\_goal}, \mathsf{update\_init})$, where (1) $\mathcal{T}_0 = (G_0, \eta_0, X_u)$ is the* base task *with initial goal states $G_0 \subseteq S$, initial state distribution η_0 and unsafe states X_u, (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 reach-avoid tasks $\mathcal{T}_i = (G_i, \eta_i, X_u)$ for each $i \in \mathbb{N}_0$, where $\mathcal{T}_0$ is defined as above and $\mathcal{T}_{i+1} = (\mathsf{update\_goal}(G_i), \mathsf{update\_init}(\eta_i), X_u)$ for $i \in \mathbb{N}_0$.*

For each task to be satisfiable, we assume that the goal region and the support of the initial distribution of each task are contained within the of safe sets X_s, i.e. that $G_i \subseteq X_s$ and $\mathrm{supp}(\eta_i) \subseteq X_s$ hold for all $i \in \mathbb{N}_0$.

3.2 Certificates of Correct Generalization

We now define our certificates for validating generalizability in inductive reach-avoid tasks. Consider an inductive reach-avoid task $\mathcal{T}$ giving rise to a sequence of reach-avoid tasks $\mathcal{T}_0, \mathcal{T}_1, \ldots$. Our certificate is defined with respect to a set $\xi = \{(\mathcal{T}_0, \zeta_0), (\mathcal{T}_1, \zeta_1), \ldots\}$ of task-trajectory pairs, where each ζ_i is a finite (possibly empty) set of MDP trajectories. These trajectories serve as *demonstrations* of what a trajectory that satisfies the reach-avoid task $\mathcal{T}_i$ should look like.

Before defining the certificate, we fix some additional notation. For each task index $i \in \mathbb{N}_0$, let $\zeta_i = \{\zeta_i^1, \ldots, \zeta_i^{K_i}\}$ be the set of demonstration trajectories for task $\mathcal{T}_i$ with $K_i \in \mathbb{N}_0$ and each $\zeta_i^k = (s_{i,j}^k)_{j=0}^{\infty}$ being an MDP trajectory. We use $T_{G_i}(\zeta_i^k) = \inf\{j \in \mathbb{N}_0 \mid s_{i,j}^k \in G_i\}$ to denote the index of the first state along trajectory ζ_i^k that is in the goal region of the task $\mathcal{T}_i$, with $T_i = \infty$ if the goal region is not reached.

Formally, a certificate of correct generalization is a function $\mathcal{C} : S \times \mathbb{N}_0 \to \mathbb{R}$ assigning a real value to each MDP state and task index pair. The certificate value is required to be (1) non-negative at all safe states along the demonstration trajectories, (2) strictly decreasing along each demonstrations trajectory until they reach a goal state, (3) decreasing across subsequent tasks, and (4) negative at all unsafe states.

Definition 2 (Certificates of correct generalization)**.** *Consider an inductive reach-avoid task $\mathcal{T}$ and a set $\xi = \{(\mathcal{T}_0, \zeta_0), (\mathcal{T}_1, \zeta_1), \ldots\}$ of task-trajectory pairs defined as above. A* certificate of correct generalization *for the set of demonstration trajectories ξ is a function $\mathcal{C} : S \times \mathbb{N}_0 \to \mathbb{R}$, such that the following conditions hold:*

1. *(Non-Negativity at Safe States) For each task index $i \in \mathbb{N}_0$, trajectory ζ_i^k and safe state $s_{i,j}^k \in X_s$ along the trajectory ζ_i^k, we have $\mathcal{C}(s_{i,j}^k, i) \geq 0$.*
2. *(Strict Decrease Within a Task Until the Goal) For each task index $i \in \mathbb{N}_0$, there exists $\varepsilon_i > 0$ such that for each trajectory ζ_i^k and safe state $s_{i,j}^k \in X_s$ along the trajectory ζ_i^k with $0 \leq j < T(\zeta_i^k)$, we have $\mathcal{C}(s_{i,j+1}^k, i) \geq 0$ and $\mathcal{C}(s_{i,j}^k, i) > \mathcal{C}(s_{i,j+1}^k, i) + \varepsilon_i$.*

3. *(Decrease Across Subsequent Tasks) For each task index $i \in \mathbb{N}_0$, the certificate value at the goal state of every trajectory for task $\mathcal{T}_i$ is greater than the value at the initial state of every trajectory for task $\mathcal{T}_{i+1}$, i.e.*

$$\min_{\substack{1\le k\le K_i \\ T(\zeta_i^k)<\infty}} \mathcal{C}(s^k_{i,T(\zeta_i^k)}, i) > \max_{\substack{1\le k\le K_{i+1} \\ T(\zeta_{i+1}^k)<\infty}} \mathcal{C}(s^k_{i+1,T(\zeta_{i+1}^k)}, i{+}1)$$

4. *(Strict Negativity at Unsafe States) For all unsafe states $s \in X_u$ and task indices $i \in \mathbb{N}_0$, we have $\mathcal{C}(s, i) < 0$.*

The intuition behind our certificate is that it captures and quantifies *progress* along demonstration trajectories towards satisfying their reach-avoid tasks. This is because the certificate value is required to be non-negative at safe states and to strictly decrease along a demonstration trajectory while keeping the certificate value non-negative, until the goal region is reached (conditions 1 and 2). Hence, the trajectory must eventually reach the goal region, and so the certificate value measures progress towards the goal. Moreover, as the certificate value is negative at unsafe states (condition 4), the trajectory must avoid unsafe states until the goal is reached, and so the reach-avoid task is satisfied. Finally, the certificate furthermore captures the inductive relation between subsequent task instances in the inductive reach-avoid task (condition 3).

The following theorem formalizes this intuition, and shows that if a certificate of correct generalization exists, then all demonstration trajectories satisfy their reach-avoid tasks. Hence, our certificate indeed captures and quantifies progress along demonstration trajectories towards satisfying their reach-avoid tasks. This will allow us to utilize our certificates to evaluate generalization properties of RL algorithms, by checking whether they preserve this progress behaviour along the trajectories generated for new and unseen tasks.

Theorem 3. *Let $\mathcal{T}$ be an inductive reach-avoid task giving rise to a sequence of reach-avoid tasks $\mathcal{T}_0, \mathcal{T}_1, \cdots$, and let $\xi = \{(\mathcal{T}_0, \zeta_0), (\mathcal{T}_1, \zeta_1), \ldots\}$ be a set of task-trajectory pairs. Then a certificate of correct generalization for the set of demonstration trajectories ξ exists then $\zeta_i^k \models \mathcal{T}_i$ for all task indices $i \in \mathbb{N}_0$ and trajectory indices $1 \le k \le K_i$.*

Proof. Suppose that $\mathcal{C}$ is a certificate of correct generalization for ξ. We need to show that, for each task index $i \in \mathbb{N}_0$ and trajectory index $1 \le k \le K_i$, we have $\zeta_i^k \models \mathcal{T}_i$. That is, if $\zeta_i^k = (s_{i,j}^k)_{j=0}^{\infty}$ is an MDP trajectory, we need to show that there exists an index $T \in \mathbb{N}_0$ such that $s_{i,T}^k \in G_i$ and $s_{i,j}^k \notin X_u$ for $0 \le j < T$.

The existence of an index $T \in \mathbb{N}_0$ such that $s_{i,T}^k \in G_i$ follows by conditions 1 and 2 in Definition 2. This is because the support of the initial state distribution η_i of task $\mathcal{T}_i$ is assumed to be contained within the safe set X_s (see Section 3.1). Hence, by condition 1, we have $\mathcal{C}(s_{i,0}^k, i) \ge 0$. On the other hand, by condition 2, it follows that until a state in G_i is reached, we will have $\mathcal{C}(s_{i,j}^k, i) \ge \mathcal{C}(s_{i,j+1}^k, i) + \varepsilon_i$ and $\mathcal{C}(s_{i,j+1}^k, i) \ge 0$. Hence, a state in G_i is reached in at most $T \le \lceil \mathcal{C}(s_{i,0}^k, i)/\varepsilon_i \rceil$ steps. Moreover, since $\mathcal{C}(s_{i,j}^k, i) \ge 0$ holds for $0 \le j \le T$, it follows by condition 4 that $s_{i,j}^k \notin X_u$ for $0 \le j < T$. This concludes the proof that $\zeta_i^k \models \mathcal{T}_i$. □

3.3 Evaluating RL Generalizability via Certificates

We now show how our certificates of correct generalization can serve as a tool for evaluating and comparing the generalization capabilities of different RL algorithms $\pi_1, \ldots, \pi_n$.

First, our framework learns a certificate of correct generalization for a given set of demonstration trajectories $\xi = \{(\mathcal{T}_0, \zeta_0), (\mathcal{T}_1, \zeta_1), \ldots\}$. The training procedure for learning certificates and the details on how demonstration trajectories are collected are presented in Section 4.

Second, a finite number of test reach-avoid tasks $\Phi^{\text{test}} \subseteq \mathcal{T}$ are chosen to assess generalizability of given RL agents $\pi_1, \ldots, \pi_n$. We require that demonstration trajectories for test reach-avoid tasks were not used in the certificate training procedure. That is, a reach-avoid task $\mathcal{T}_i$ can be used as a test task only if the set of demonstration trajectories ζ_i is empty.

Finally, for each RL agent π_i and for each test reach-avoid task $\phi \in \Phi^{\text{test}}$, we use π_i to generate a trajectory $\xi_{\pi_i}^{\phi}$ for the test task ϕ. We then count the total number of violations of the certificate conditions along $\xi_{\pi_i}^{\phi}$. That is, for each condition 1-4 in Definition 2, we count at how many states along the trajectory $\xi_{\pi_i}^{\phi}$ the condition is violated. In reach-avoid tasks, all steps beyond the first unsafe state (denoted as "OC") are treated as violations. We denote by $N_{\pi_i}^{\phi}$ the total number of certificate violations for task ϕ, and define

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

to be the *total number of certificate violations* of the RL agent π_i across all test tasks in Φ^{test}. The *percentage of certificate violations* is computed using the ratio of the number of certificate violations vs. the total number of states in the trajectory.

We then compare the percentages of certificate violations for different RL agents $\pi_1, \ldots, \pi_n$, and conclude that RL agents π_i with the lower percentages of certificate violations have better generalizability properties.

This framework provides a principled method to evaluate generalization properties of RL agents on unseen tasks, by using certificates of correct generalization from given demonstrations as a validation tool. By comparing the percentages of certificate violations across different RL algorithms, we gain insights into their ability to generalize to new tasks and preserve the progress behaviour captured by the certificate.

4 Learning Certificates

This section describes the training procedure for learning a certificate of correct generalization. Consider an inductive reach-avoid task $\mathcal{T}$ giving rise to a sequence of reach-avoid tasks $\mathcal{T}_0, \mathcal{T}_1, \ldots$. Given a set of demonstration trajectories $\xi = \{(\mathcal{T}_0, \zeta_0), (\mathcal{T}_1, \zeta_1), \ldots\}$ defined as in the previous section, our objective is to train a neural network to approximate the certificate function $\mathcal{C} : S \times \mathbb{N}_0 \to \mathbb{R}$ defined in Definition 2.

Training Data. We start by describing how our training data, i.e. the demonstration trajectories used to train a neural network certificate, are collected. We train the certificate on finite-length prefixes of trajectories for tasks $\mathcal{T}_0, \cdots \mathcal{T}_{n-1}$, i.e., the first n tasks in the inductive reach-avoid task $\mathcal{T}$, where $n \in \mathbb{N}$ is an algorithm parameter. We use finite-length prefixes

because one cannot obtain infinite-length trajectories in practice. We refer to tasks $\mathcal{T}_0, \cdots \mathcal{T}_{n-1}$ as the *training tasks* and tasks $\mathcal{T}_n, \mathcal{T}_{n+1} \cdots$ as the *testing tasks*. Hence, demonstration trajectories will be finite-length trajectories in the sets ζ_i for $0 \leq i \leq n-1$, whereas $\zeta_i = \emptyset$ for $i \geq n$.

For each training task $\mathcal{T}_i$ with $0 \leq i \leq n-1$, the set of demonstration trajectories ζ_i comprises of (1) positive examples, i.e. trajectories that satisfy the reach-avoid task $\mathcal{T}_i$, and (2) negative examples, i.e. trajectories that violate $\mathcal{T}_i$. Positive examples are needed in order to capture progress behaviour in trajectories that satisfy the reach-avoid task with respect to conditions 1-3 in Definition 2. Among the negative examples, all trajectories are of the kind that reach an unsafe state and thus violate the avoidance objective, which are needed to capture condition 4 in Definition 2. We do not consider finite-length trajectories that do not violate safety but violate reachability, since one cannot ascertain whether such a trajectory could have satisfied the specification if it were longer.

These training trajectories are collected using an Oracle that generates trajectories of positive and negative examples in the environment for the given reach-avoid task. Positive and negative trajectories are obtained by sampling many trajectories in the environment using a planner, and assigning them positive or negative depending on whether the trajectory satisfies the reach-avoid task.

Training Procedure. Our training procedure is formulated as a learning task, where the goal is to minimize a loss function which is designed to enforce conditions 1-4 in Definition 2 along demonstration trajectories. In particular, our loss function is a sum of four loss terms, each corresponding to one condition in Definition 2, where the outer sum is taken over all training tasks and all demonstration trajectories:

$$
\begin{aligned}
\mathsf{Loss} = \sum_{i=0}^{n-1} \sum_{(s_{i,j})_{j=0}^{l} \in \zeta_i} \Bigg(& \\
& \underbrace{+ \lambda_{\text{safe}} \sum_{j=0}^{l} \max\{0, -C(s_{i,j}, i)\} \cdot 1_{s_{i,j} \in X_s}}_{\text{Penalizes to enforce condition (1)}} \\
& \underbrace{+ \sum_{j=0}^{l-1} \max\{0, C(s_{i,j+1}, i) + \varepsilon_i - C(s_{i,j}, i)\} \cdot 1_{(s_{i,j}, s_{i,j+1}) \in X_s}}_{\text{Penalizes to enforce condition (2)}} \\
& \underbrace{+ \sum_{(s_{i+1,j})_{j=0}^{l} \in \zeta_{i+1}} \max\{0, C(s_{i+1,0}, i+1) + \varepsilon_i - C(s_{i,l}, i)\} \cdot 1_{s_{i,l} \in G_i}}_{\text{Penalizes to enforce condition (3)}} \\
& \underbrace{+ \lambda_{\text{unsafe}} \sum_{j=0}^{l} \max\{0, C(s_{i,j}, i)\} \cdot 1_{s_{i,j} \in X_u} \Bigg)}_{\text{Penalizes to enforce condition (4)}}
\end{aligned}
$$

Here, for each task index $i \in \mathbb{N}_0$ and for each finite-length trajectory in ζ_i, we use l to denote the index of the first visit to either the goal region G_i of T_i (for positive example trajectories) or the unsafe set X_u (for negative example trajectories). The first summation enforces the non-negativity condition (1) at safe states X_s by penalizing violations. The fourth summation enforces the strict negativity condition (4) at unsafe states X_u. The scaling factors λ_{safe} and λ_{unsafe} are hyperparameters that control the importance of these penalties. The second summation penalizes violations of monotonic decrease within task trajectories, enforcing the certificate function to decrease as the trajectory progresses. The third summation penalizes violations of monotonic decrease across tasks, ensuring the certificate values at the goal states of task $\mathcal{T}_i$ are greater than those of the start states of $\mathcal{T}_{i+1}$.

We use the *Long Short-Term Memory* [Hochreiter, 1997] (LSTM) architecture to train the certificate function. The LSTM acts as a function $\mathcal{C}(s, i)$ taking as input a state s and a task index i, and outputs a real number representing the certificate value. The LSTM is trained on the set of all sequences of concatenated trajectories of the form $(\zeta_0^{k_0}, \zeta_1^{k_1}, \ldots, \zeta_{n-1}^{k_{n-1}})$, where $\zeta_i^{k_i} \in \zeta_i$ is a demonstration trajectory for task $\mathcal{T}_i$ (generated by the oracle). We generate and use multiple batches of ξ for effective training We choose LSTMs since they are well-suited to capture sequential dependencies in data. Trajectories represent ordered sequences of states, where the progression along each trajectory encodes information about task dynamics, safety, and proximity to the target state. Furthermore, concatenated trajectories ensure that dynamics across subsequent tasks are encoded. Since LSTMs are designed to maintain memory and capture long-term dependencies, we expect the LSTMs to capture these intra- and inter-trajectory dependencies while respecting safety conditions.

5 Experimental Evaluation

The purpose of the empirical evaluation is to demonstrate that our certification procedure can serve as a litmus test to examine generalizability of RL algorithms.

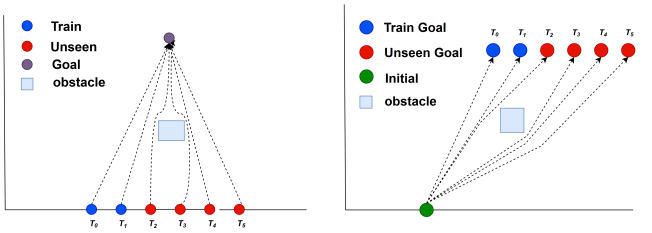
Our codebase is available at https://anonymous.4open.science/r/certificates-B6D7.

5.1 Experimental Set-up

Objective and Procedure. Our objective is to demonstrate that a lower percentage of certificate violations indicates better generalizability of an RL algorithm. We do so by comparing the results of our certificate-based evaluation to a ground truth.

Given an RL algorithm $\mathcal{A}$ and an inductive task $\mathcal{T}$, we learn a (generalizable) policy using the first n tasks $\mathcal{T}_0, \mathcal{T}_1, \cdots \mathcal{T}_{n-1}$ of the inductive task. Then, we evaluate the following two metrics to assess generalizability of the learned policy:

- (Ground Truth) We compute the *number of unseen tasks solved* by the learned policy. That is, we consider how many tasks starting from $\mathcal{T}_n, \mathcal{T}_{n+1}, \cdots$ the policy can accomplish. Larger the number of unseen tasks the policy can solve, the better its generalizability.
- (Certificate-Guided) We use our framework in Section 3.3 to compute the percentage of certificate violations. The lower the percentage, the better its generalizability.



(a) Moving initial location and stationary goal position with obstacle

(b) Stationary initial location and moving goal position with obstacle

Figure 2. Illustrations of inductive tasks in Car Environment

By demonstrating a strong correlation between the two metrics, we will confirm the effectiveness of our framework in assessing generalization.

To ensure fairness, we train all RL algorithms on the same set of tasks from the family of inductive task. In the ground truth evaluation, all RL algorithms are tested on the same set of unseen tasks. For each inductive task, we train the certificate once and use the same certificate to evaluate all RL algorithms.

RL Algorithms. We evaluate a diverse set of state-of-the-art generalizable RL algorithms that have the ability to generalize to similar but different tasks:

- Standard (Non-Generalizable) RL Algorithms, such as ARS and its variants (ARS + Reward Aggregation and ARS + Goal Conditioning) and PPO [Schulman *et al.*, 2017], which serve as strong baselines despite lacking explicit generalization mechanisms.
- Generalizable RL algorithms across three categories:
  - *Inductive Generalization algorithms* PSMP [Inala *et al.*, 2020] and GenRL-ARS [Subramanian *et al.*, 2024], which leverage inductive structures between tasks;
  - *Meta-Learning algorithms* MAML-Reinforce [Finn *et al.*, 2017] and VariBAD-A2C [Zintgraf *et al.*, 2021], enabling rapid adaptation through task-specific representations;
  - *Goal-Conditioned algorithms* C-Learning [Naderian *et al.*, 2021] and HER-DDPG [Andrychowicz *et al.*, 2017], which specialize in goal-conditioned generalization.

Environments and Inductive Tasks. In the interest of space, we discuss our results on two environments: 1) **Car** environment [Inala *et al.*, 2020] involves moving a rectangular car from an initial to goal location in a continuous 2D plane, and 2) **Fetch-Reach** environment [Brockman *et al.*, 2016] involves a 7-DOF robotic arm moving its end effector from an initial position to a goal position in a continuous 3D space.

Illustrations of inductive reach-avoid tasks for the Car Environment are given in Figure 1a and Figure 2. The inductive reach-avoid task for the Fetch-Reach environment is to navigate from a moving initial location to a stationary goal location, similar to Figure 2a. We also consider the *reachability* variant of all these tasks where the obstacle is absent.

Full descriptions of environments, tasks, and results on more environments are presented in the Appendix.

5.2 Observations

The results are presented using plots that display the percentage of certificate violations and the number of successfully solved tasks for various RL algorithms (see Figure 3). Each marker shows the mean percentage of violations over 10 runs with error bars indicating the variance. The number of solved tasks (eg. $2/15$) is annotated alongside the markers, providing ground truth evidence for the correctness of the certificate. This means that the algorithm generalized to 2 out of 15 unseen tasks. High violations correspond to poor generalization guaranteed by low number of solved tasks while low violations indicate better generalizability guaranteed by high number of solved tasks. For reach-avoid tasks, "OC" (Obstacle Collision) is used to indicate when a trajectory fails by colliding with an obstacle, with all steps after the collision treated as violations.

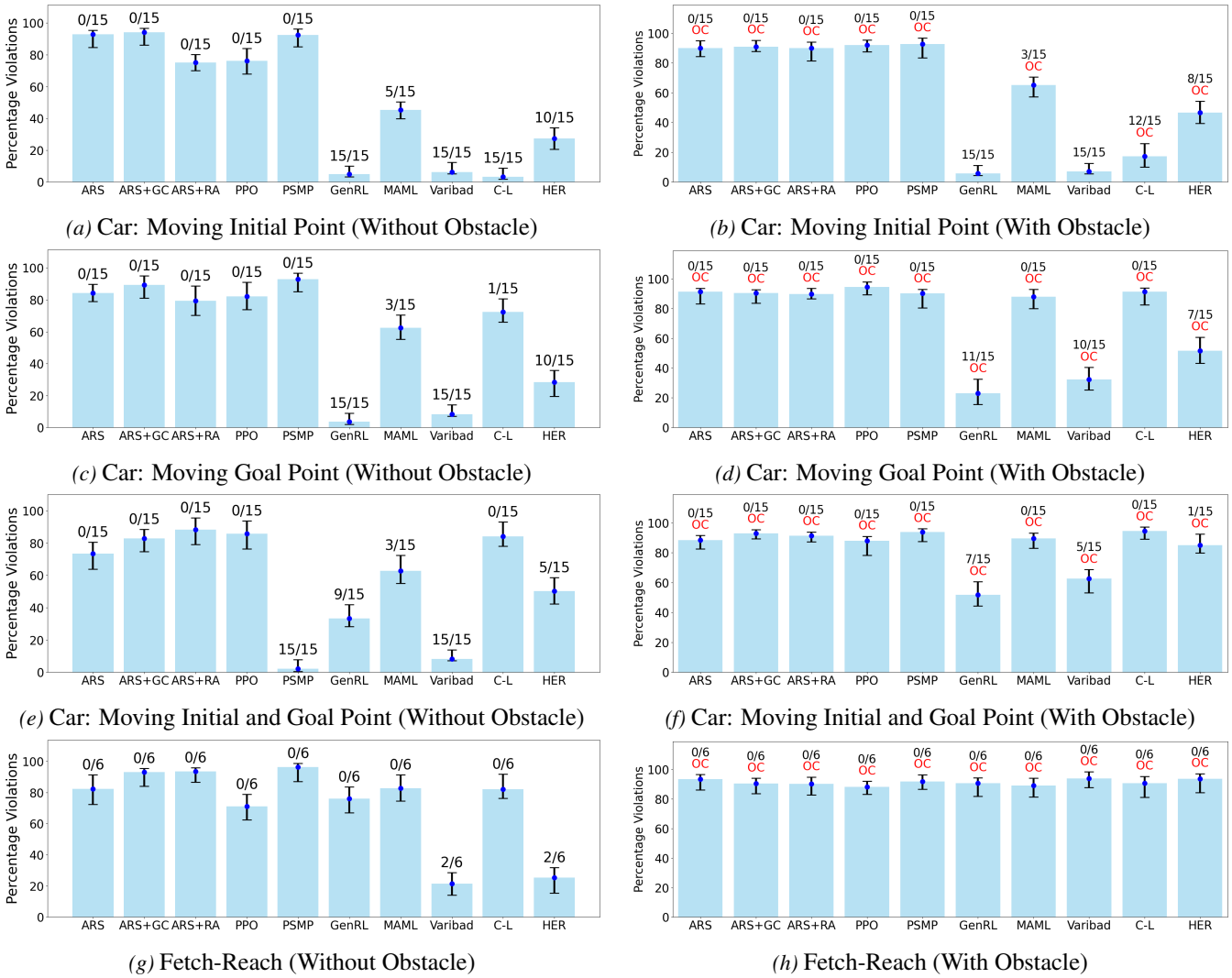
Certificate violations are a good measure of generalizability. All our experiments show that a lower percentage of certificate violations correlates to a higher number of successfully solved unseen tasks, hence indicating better generalizability properties of RL algorithms. Similarly, higher percentage of certificate violations correlates to a lower number of successfully solved unseen tasks. Our evaluation results also align with priors one may have w.r.t. algorithm performance. For instance, standard (non-generalizable) RL approaches ARS and PPO perform poorly in all experiments.

Reach-avoid tasks are hard generalization benchmarks. Most RL algorithms performed poorly on reach-avoid tasks, as shown by high certificate violations and low solved task counts. In the high dimensional obstacle environments such as Car and Fetch, even generalizable RL algorithms struggled, with some runs showing over 90% violations. This illustrates that obstacles create a significant challenge for RL generalization. To analyze further, we tested in low-dimensional obstacle environments, such as Car2D, and observed similarly poor performance (Results in Appendix 5). This highlights that obstacles are inherently difficult constraints, even in simpler settings, underscoring their role as a strict-constraint for generalization in locomotion-based navigation tasks.

But, in contrast to reach-avoid tasks (tasks with obstacles), RL algorithms in environments without obstacles demonstrated better performance. Generalizable algorithms, such as GenRL and VariBAD, consistently achieved low violations. This underscores the relative simplicity of reachability tasks compared to reach-avoid tasks, where obstacle avoidance is not a factor and algorithms without explicit generalization capabilities perform poorly regardless of the task.

6 Conclusion

This work presents a novel framework for evaluating and comparing the generalization capabilities of RL agents. Our framework is based on the novel notion of certificates of correct generalization, which we use to provide a method for assessing 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 our framework.



(a) Car: Moving Initial Point (Without Obstacle)

(b) Car: Moving Initial Point (With Obstacle)

(c) Car: Moving Goal Point (Without Obstacle)

(d) Car: Moving Goal Point (With Obstacle)

(e) Car: Moving Initial and Goal Point (Without Obstacle)

(f) Car: Moving Initial and Goal Point (With Obstacle)

(g) Fetch-Reach (Without Obstacle)

(h) Fetch-Reach (With Obstacle)

Figure 3. Performance evaluation of tasks across different environments. (a)-(h) present the evaluation results, showing the percentage of certificate violations and the number of successfully solved tasks for various RL algorithms. Each marker represents the mean percentage of violations over 10 runs, with error bars indicating the range (minimum to maximum). The numbers annotated next to each marker (e.g., $0/15$) denote the number of successfully solved tasks out of the total test tasks. Higher violations correspond to poor generalization, while a lower percentage of violations indicates better generalization.

References

Alessandro Abate, Daniele Ahmed, Mirco Giacobbe, and Andrea Peruffo. Formal synthesis of lyapunov neural networks. *IEEE Control Systems Letters*, 5(3):773–778, 2021.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in Neural Information Processing Systems*, 30, 2017.

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural lyapunov control. In *NeurIPS*, pages 3240–3249, 2019.

Krishnendu Chatterjee, Thomas A. Henzinger, Mathias Lechner, and Dorde Zikelic. A learner-verifier framework for neural network controllers and certificates of stochastic systems. In *TACAS*, pages 3–25, 2023.

Charles Dawson, Sicun Gao, and Chuchu Fan. Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control. *IEEE Transactions on Robotics*, 39(3):1749–1767, 2023.

Alec Edwards, Andrea Peruffo, and Alessandro Abate. Fossil 2.0: Formal certificate synthesis for the verification and control of dynamical models. In *ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 26:1–26:10, 2024.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pages 1126–1135, 2017.

Sepp Hochreiter. Long short-term memory. *Neural Computation*, 1997.

Jeevana Priya Inala, Osbert Bastani, Zenna Tavares, and Armando Solar-Lezama. Synthesizing programmatic policies that inductively generalize. In *International Conference on Learning Representations*, 2020.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

Ezgi Korkmaz. A survey analyzing generalization in deep reinforcement learning. *arXiv preprint arXiv:2401.02349*, 2024.

Mathias Lechner, Dorde Zikelic, Krishnendu Chatterjee, and Thomas A. Henzinger. Stability verification in stochastic control systems via neural network supermartingales. In *AAAI*, pages 7326–7336, 2022.

Dhruv Malik, Yuanzhi Li, and Pradeep Ravikumar. When is generalizable reinforcement learning tractable? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Frederik Baymler Mathiesen, Simeon C. Calvert, and Luca Laurenti. Safety certification for stochastic systems via neural barrier functions. *IEEE Control Systems Letters*, 7:973–978, 2023.

Panteha Naderian, Gabriel Loaiza-Ganem, Harry J. Braviner, Anthony L. Caterini, Jesse C. Cresswell, Tong Li, and Animesh Garg. C-learning: Horizon-aware cumulative accessibility estimation. *International Conference on Learning Representations*, 2021.

Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: continual adaptation for model-based rl. *International Conference on Learning Representations*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Vignesh Subramanian, Rohit Kushwah, Subhajit Roy, and Suguman Bansal. Inductive generalization in reinforcement learning from specifications. *arXiv preprint arXiv:2406.03651*, 2024.

Hongchao Zhang, Junlin Wu, Yevgeniy Vorobeychik, and Andrew Clark. Exact verification of relu neural control barrier functions. In *NeurIPS*, 2023.

Dorde Zikelic, Mathias Lechner, Thomas A. Henzinger, and Krishnendu Chatterjee. Learning control policies for stochastic systems with reach-avoid guarantees. In *AAAI*, pages 11926–11935, 2023.

Luisa Zintgraf, Sebastian Schulze, Cong Lu, Leo Feng, Maximilian Igl, Kyriacos Shiarlis, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: Variational bayes-adaptive deep rl via meta-learning. *Journal of Machine Learning Research*, 22(289):1–39, 2021.

Ethical Statement

This research develops algorithms and benchmarks for evaluating reinforcement learning generalization. Our work does not involve human subjects, sensitive data, or direct real-world deployment. While our methods aim to improve AI robustness and safety, we acknowledge potential dual-use concerns if applied to autonomous systems without appropriate safeguards. All code and data will be open-sourced to support reproducibility and responsible development.

Appendix

1 Training Details

The experimental setup includes the following details:

- **Number of Training Tasks:** 5 tasks for Car and Car2D environments, and 4 tasks for Reacher and Fetch environments.
- **Number of Unseen Tasks:** 15 tasks for Car and Car2D environments, and 6 tasks for Reacher and Fetch environments.
- **Number of Repetitions of the Experiments:** 10.
- **Training Compute:** SLURM cluster running Intel Xeon Gold 6226 CPUs, operating at 2.7 GHz with 24 cores per node. Each node is equipped with 192GB of RAM.

Learning hyperparameters for our framework is given in Table 1 and 2.

2 Experimental Setup

Our approach is evaluated across several environments, including the Car, Car2D, 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 environment** has a 4-dimensional state space, including x and y positions, orientation, and velocity, with 2 actions: velocity and steering direction. The **Car2D environment** is a simplified 2-dimensional state space and observation space, consisting of the x and y coordinates of the agent, with 2 actions: velocity and direction. The **OpenAI Fetch-Reach environment** involves a robotic arm with an 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 & Car2D Environment** (Fig. 1):
  - Setting 1 the initial distribution is updated by shifting the initial position along the x-axis by a fixed value C, i.e.,

$$\eta_{i+1}(s) = \eta_i(s + (C, 0)).$$
$$G_{i+1} = G_i + (0, 0).$$

  - Setting 2: the goal is updated by shifting the target position along the x-axis by C, i.e.,

$$\eta_{i+1}(s) = \eta_i(s + (0, 0)).$$
$$G_{i+1} = G_i + (C, 0).$$

  - Setting 3: both the initial and target positions along the x-axis by C, combining both updates.

$$\eta_{i+1}(s) = \eta_i(s + (C, 0)).$$
$$G_{i+1} = G_i + (C, 0).$$

- **Reacher two arm robot environment** (Fig. 4a): 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** : 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.

3 Algorithms and Hyperparameters

The evaluated algorithms are categorized as follows:

- **Standard RL Algorithms: ARS**, **ARS+GC**, **ARS+RA**, and **PPO** are baseline reinforcement learning methods known for their effectiveness in diverse tasks. These algorithms lack explicit mechanisms for generalization, focusing instead on optimizing performance in specific tasks.
- **Inductive Generalization Algorithms: PSMP** and **GenRL** leverage inductive biases to generalize across tasks with structural similarities. These methods enhance transferability by exploiting shared properties within task families.
- **Meta-Learning Algorithms: MAML** and **Varibad** are designed for rapid adaptation to new tasks with minimal training. **MAML** employs gradient-based meta-learning, while **Varibad** uses variational inference for task embeddings, enabling zero-shot generalization.
- **Goal-Conditioned RL Algorithms: C-Learning** and **HER** explicitly incorporate goals into policy learning, optimizing for tasks requiring specific goal states. These algorithms are well-suited for environments with dynamic or goal-driven objectives.

Hyperparameters used for these algorithms are given from Table 3 to 10

Algorithm 1 Learning Reach-Avoid Certificates

Input: Inductive task family $\{\mathcal{T}_i\}_{i=0}^{N}$ with goal sets $\{G_i\}$ and initial state distributions $\{\eta_i\}$.
Training tasks: $\mathcal{T}_i$ for $i = 0, \ldots, n-1$.
Testing tasks: $\mathcal{T}_i$ for $i = n, \ldots, N$.
Output: Trained certificate function $\mathcal{C}(s, i)$.

1: Generate oracle trajectories $\zeta_i = (s_{i,0}, s_{i,1}, \ldots, s_{i,l_i})$ for training tasks $\mathcal{T}_i$ $(i = 0, \ldots, n)$ to solve their respective goal sets G_i.
2: Initialize LSTM model $\mathcal{C}(s, i)$ with random weights.
3: **while** the model has not converged **do**
4: Compute the loss:

$$
\begin{aligned}
\mathsf{Loss} = \sum_{i=0}^{n-1} \sum_{(s_{i,j})_{j=0}^{l} \in \zeta_i} \Bigg(& \\
& \underbrace{+ \lambda_{\text{safe}} \sum_{j=0}^{l} \max\{0, -C(s_{i,j}, i)\} \cdot 1_{s_{i,j} \in X_s}}_{\text{Penalizes to enforce condition (1)}} \\
& \underbrace{+ \sum_{j=0}^{l-1} \max\{0, C(s_{i,j+1}, i) + \varepsilon_i - C(s_{i,j}, i)\} \cdot 1_{(s_{i,j}, s_{i,j+1}) \in X_s}}_{\text{Penalizes to enforce condition (2)}} \\
& \underbrace{+ \sum_{(s_{i+1,j})_{j=0}^{l} \in \zeta_{i+1}} \max\{0, C(s_{i+1,0}, i+1) + \varepsilon_i - C(s_{i,l}, i)\} \cdot 1_{s_{i,l} \in G_i}}_{\text{Penalizes to enforce condition (3)}} \\
& \underbrace{+ \lambda_{\text{unsafe}} \sum_{j=0}^{l} \max\{0, C(s_{i,j}, i)\} \cdot 1_{s_{i,j} \in X_u} \Bigg)}_{\text{Penalizes to enforce condition (4)}}
\end{aligned}
$$

5: Perform backpropagation and update the weights of $\mathcal{C}(s, i)$.
6: **return** Trained certificate function $\mathcal{C}(s, i)$.

Hyperparameter	**Car-Parking**	**Reacher**	**Fetch-Reach**
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Batch Size	64	64	128
Sequence Length	20	20	50
Dropout Rate	0.2	0.2	0.2
Optimizer	Adam	Adam	Adam
Gradient Clipping	± 0.5	± 0.5	± 0.5
Network Architecture	2 hidden LSTM layers, 32 nodes (`tanh`); output layer (`relu`)	2 hidden LSTM layers, 32 nodes (`tanh`); output layer (`relu`)	3 hidden LSTM layers, 64 nodes (`tanh`); output layer (`relu`)

Table 1. Hyperparameters for the LSTM-based Reachability Certificate Function

Hyperparameter	**Car-Parking**	**Reacher**	**Fetch-Reach**
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Batch Size	64	64	128
Sequence Length	20	20	50
Dropout Rate	0.2	0.2	0.2
Optimizer	Adam	Adam	Adam
Gradient Clipping	± 0.5	± 0.5	± 0.5
Network Architecture	2 hidden LSTM layers, 32 nodes (`tanh`); output layer (no activation)	2 hidden LSTM layers, 32 nodes (`tanh`); output layer (no activation)	3 hidden LSTM layers, 64 nodes (`tanh`); output layer (no activation)

Table 2. Hyperparameters for the LSTM-based Reach-Avoid Certificate Function

Hyperparameter	**Car & Car2D**	**Reacher**	**Fetch-Reach**
Epsiode Length	30	30	100
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Number of Directions Sampled	30	30	30
Number of Top Samples Used	8	8	8
Network Architecture	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	3 hidden layers, 64 nodes (`ReLU`), output layer (`tanh`)

Table 3. Hyperparameters for ARS, ARS + GC, ARS + RA

Hyperparameter	Car & Car2D	Reacher	Fetch-Reach
Epsiode Length	30	30	100
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Batch Size	64	64	256
Discount Factor	0.99	0.99	0.99
GAE Parameter	0.95	0.95	0.95
Clipping Parameter	0.2	0.2	0.2
Value Loss Coefficient	0.5	0.5	0.5
Entropy Coefficient	0.01	0.01	0.01
Replay Buffer Size	10^4	10^4	10^4
Network Architecture	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	3 hidden layers, 64 nodes (`ReLU`), output layer (`tanh`)

Table 4. Hyperparameters for PPO

Hyperparameter	Car & Car2D	Reacher	Fetch-Reach
Epsiode Length	30	30	100
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Batch Size	64	64	256
Maximum Number of Modes	2	2	2
λ	100	100	100
Student Action Function Grammar	Proportional	Proportional	Proportional
Switching Condition Grammar	Boolean tree (depth 1 or 2)	Boolean tree (depth 1 or 2)	Boolean tree (depth 1 or 2)
Sampling for Teacher Initialization	10 top trajectories (CEM)	10 top trajectories (CEM)	10 top trajectories (CEM)
Network Architecture	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	3 hidden layers, 64 nodes (`ReLU`), output layer (`tanh`)

Table 5. Hyperparameters for PSMP

Hyperparameter	Car & Car2D	Reacher	Fetch-Reach
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Number of Directions Sampled	30	30	30
Number of Top Samples Used	8	8	8
Reward Aggregation Scheme	softmin	softmin	softmin
Batch Size	64	64	256
Network Architecture	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	3 hidden layers, 64 nodes (`ReLU`), output layer (`tanh`)

Table 6. Hyperparameters for GenRL - ARS

Hyperparameter	Car & Car2D	Reacher	Fetch-Reach
Epsiode Length	30	30	100
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Batch Size	64	64	256
Number of Meta-Tasks	4	5	5
Number of Inner Loop Steps	0	0	0
Discount Factor	0.99	0.99	0.99
Entropy Coefficient	0.01	0.01	0.01
Policy Network Architecture	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	3 hidden layers, 64 nodes (`ReLU`), output layer (`tanh`)
Optimizer	Adam	Adam	Adam
Gradient Clipping	± 0.5	± 0.5	± 0.5
KL-Divergence Regularization	Enabled, coefficient = 0.1	Enabled, coefficient = 0.1	Enabled, coefficient = 0.1
Trajectory Length	200 steps	200 steps	200 steps
Meta-Batch Size	4 tasks	5 tasks	5 tasks

Table 7. Hyperparameters for MAML-Reinforce

Hyperparameter	Car & Car2D	Reacher	Fetch-Reach
Epsiode Length	30	30	100
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Batch Size	64	64	256
Epsilon	1×10^{-5}	1×10^{-5}	1×10^{-5}
Discount Factor	0.95	0.95	0.95
Max Gradient Norm	0.5	0.5	0.5
Value Loss Coefficient	0.5	0.5	0.5
Entropy Coefficient	0.01	0.01	0.01
GAE Parameter	0.95	0.95	0.95
ELBO Loss Coefficient	1.0	1.0	1.0
Task Embedding Size	5	5	5
Network Architecture	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	3 hidden layers, 64 nodes (`ReLU`), output layer (`tanh`)
Encoder Architecture	Fully connected (40 nodes), GRU (hidden size 64), output layer (10 outputs, `ReLU`)	Fully connected (40 nodes), GRU (hidden size 64), output layer (10 outputs, `ReLU`)	Fully connected (40 nodes), GRU (hidden size 64), output layer (10 outputs, `ReLU`)
Reward Decoder Architecture	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	3 hidden layers, 64 nodes (`ReLU`), output layer (`tanh`)
Decoder Loss Function	Binary Cross Entropy	Binary Cross Entropy	Binary Cross Entropy

Table 8. Hyperparameters for Varibad - A2C

Hyperparameter	Car & Car2D	Reacher	Fetch-Reach
Epsiode Length	30	30	100
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Batch Size	64	64	256
Gradient Steps per Episode	64	64	64
Network Architecture	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	3 hidden layers, 64 nodes (`ReLU`), output layer (`tanh`)
Target Network Update Frequency	Every 10 steps	Every 10 steps	Every 10 steps
Discount Factor	0.9	0.9	0.9
κ	3	3	3

Table 9. Hyperparameters for C-Learning

Hyperparameter	Car & Car2D	Reacher	Fetch-Reach
Epsiode Length	30	30	100
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Critic Learning Rate	1×10^{-3}	1×10^{-3}	1×10^{-3}
Batch Size	64	64	256
Discount Factor	0.98	0.98	0.98
Replay Buffer Size	10^4	10^4	10^4
Policy Noise	0.2	0.2	0.2
Target Network Update Rate	0.005	0.005	0.005
Hindsight Strategy	Future strategy (relabel goals)	Future strategy (relabel goals)	Future strategy (relabel goals)
Hindsight Fraction	0.8	0.8	0.8
Network Architecture	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	2 hidden layers, 32 nodes (`ReLU`), output layer (`tanh`)	3 hidden layers, 64 nodes (`ReLU`), output layer (`tanh`)

Table 10. Hyperparameters for HER-DDPG

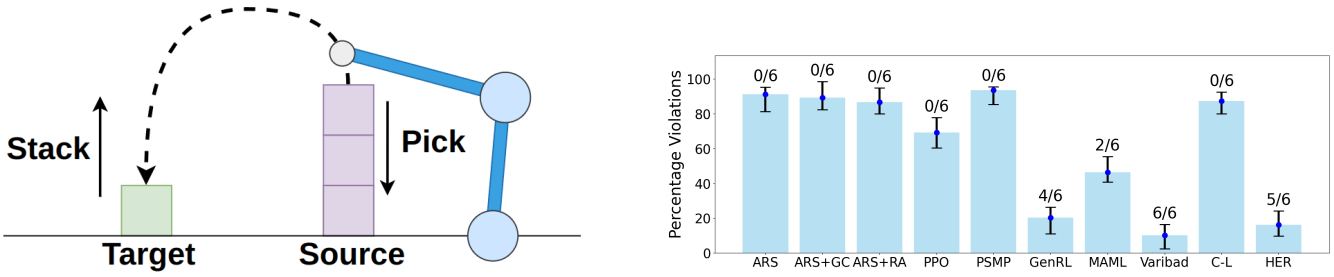
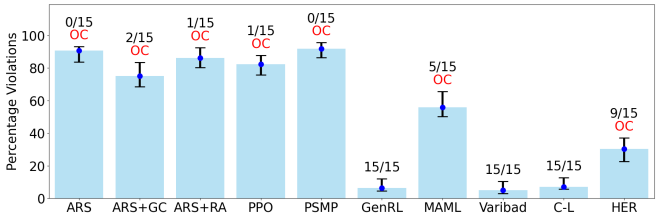
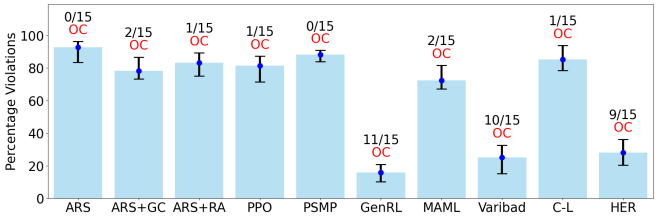


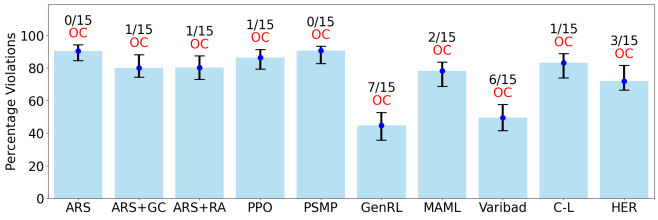
Figure 4. Performance evaluation of tasks on the Reacher 2 Arm robot environment. (a) provides the illustration of the task (b) presents the evaluation results, showing the percentage of certificate violations and the number of successfully solved tasks for various RL algorithms. Each marker represents the mean percentage of violations over 10 runs, with error bars indicating the range (minimum to maximum). The numbers annotated next to each marker (e.g., $0/6$) denote the number of successfully solved tasks out of the total test tasks. Higher violations correspond to poor generalization, while a lower percentage of violations indicates better generalization.



(a) Car2D: Moving Initial Point with obstacle



(b) Car2D: Moving Goal Point with obstacle



(c) Car2D: Moving Initial and Goal Point with obstacle

Figure 5. Performance evaluation of tasks on the Simpler Car2D environment. (a)-(c) presents the evaluation results, showing the percentage of certificate violations and the number of successfully solved tasks for various RL algorithms. Each marker represents the mean percentage of violations over 10 runs, with error bars indicating the range (minimum to maximum). The numbers annotated next to each marker (e.g., $0/15$) denote the number of successfully solved tasks out of the total test tasks. Higher violations correspond to poor generalization, while a lower percentage of violations indicates better generalization.