# Decoupled Behavioral Cloning for Scalable Inductive Generalization in RL from Specifications

**Vignesh Subramanian**[1], **Subhajit Roy**[2], **Suguman Bansal**[1]

[1]School of Computer Science, Georgia Institute of Technology, USA

[2]Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

[1]{vignesh,suguman}@gatech.edu, [2]subhajit@cse.iitk.ac.in

## Abstract

Inductive generalization is a framework for reinforcement learning (RL) generalization in which inductively related task instances admit inductively related policies. Prior work captures this structure via a higher-order *policy-evolution function* learned directly with RL, but suffers from poor *training scalability*: as training tasks grow, aggregated reward feedback becomes noisy and conflicting, destabilizing training and weakening generalization. We propose DIBS, a *decoupled behavioral cloning* approach that separates learning task-specific policies from learning the evolution function. We first learn individual teacher policies per task via standard RL, then fit the evolution function via behavioral cloning on teacher-labeled state-action pairs. This replaces noisy reward aggregation with dense, stable supervision. DIBS achieves significant improvements in both training stability and zero-shot generalization against existing RL and meta-RL algorithms.

## 1 Introduction

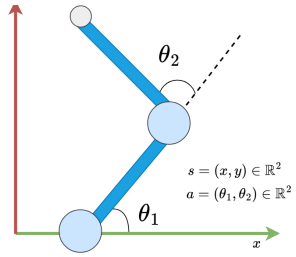
The *generalization* problem in *reinforcement learning (RL)* [33] asks how agents can learn policies that transfer to new, unseen situations [20, 7, 41, 8]. Despite remarkable successes, generalization remains a fundamental challenge. RL agents that perform well on individual tasks often fail on even slightly modified scenarios.

*Inductive generalization* (GenRL) [32] is a notion of generalization that leverages *inductive similarities* between tasks. The core intuition is that inductively related tasks must have inductively related policies. Inductive relationships arise naturally whenever tasks exhibit recursion or iteration: once we understand how to transform the policy for the $i-th$ task to the $i+1$ task by training on a few tasks, we can generalize to arbitrarily many tasks. This structure is evident in robotics and control, where physical constraints impose regular, repeating patterns. Prior work formalizes this intuition by encoding *inductive families of tasks* using *temporal logic specifications*, where tasks share identical logical structure and differ only in the instantiation of low-level predicates.

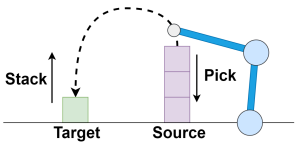
Consider the robotic arm in Fig. 1. It has to displace a tower of blocks from a source to a target. This complex task decomposes into a series of inductively related subtasks. In the $i$-th subtask, the robot relocates the topmost block from the source at height $i$ to the top of the target at height $(h-i)$, where $h$ is the initial height of the source. The 0-th task is the base task, and successive tasks are obtained by updates to the locations of the topmost blocks in the source and target.

The inductive generalization algorithm GenRL captures this structure by learning a higher-order *policy-evolution function* $\Omega : \Pi \to \Pi$ such that $\pi_{i+1} = \Omega(\pi_i)$, approximated as an $m$-degree polynomial over policy parameters. Learning reduces to inferring a small set of $\kappa$-coefficients via an RL-style training loop over training tasks (Fig. 1.c). Despite stronger generalization than several state-of-the-art methods, GenRL suffers from a fundamental *training scalability* bottleneck: policy

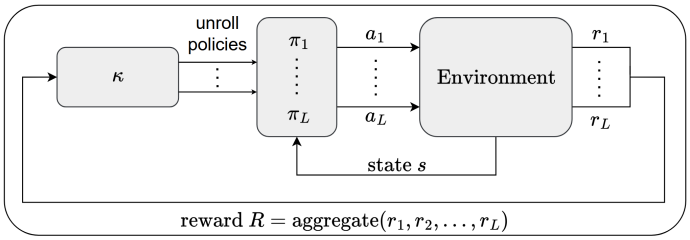
Preprint.



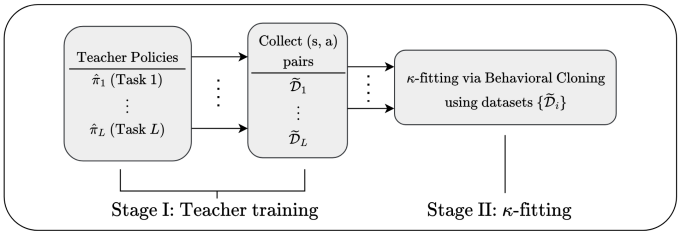

(a) Reacher benchmark environment




(b) Inductive task family in the Reacher benchmark




(c) GenRL flow diagram




(d) DIBS flow diagram

Figure 1: **Left column: Benchmark illustrations.** **(a)** Reacher benchmark environment. **(b)** Inductive task family in the Reacher benchmark. **Right column: Contrasting flow diagrams. (c)** GenRL flow diagram. **(d)** DIBS flow diagram.

learning for training tasks is tightly coupled with $\kappa$-coefficient training. As the number of training tasks grows, aggregated reward feedback becomes noisy and conflicting, making the training loop increasingly difficult to stabilize, ultimately constraining its generalization capability.

We propose DIBS, a *decoupled behavioral cloning* approach that resolves this bottleneck by separating two phases that GenRL conflates (Fig. 1.d). In **Stage I**, we independently learn *teacher policies* for each training task using standard RL. In **Stage II**, we fit the $\kappa$-template via *behavioral cloning* on teacher-labeled state-action datasets [36, 21, 29], replacing noisy multi-task reward aggregation with stable, dense supervision. Since teacher policies are fixed during $\kappa$-training, the destabilizing feedback loop in GenRL is eliminated. A further advantage is that teacher policies can be learned using *any* RL algorithm, unlike GenRL, which is tightly coupled to ARS [26], fundamentally limiting the environments it can handle. This decoupling raises non-trivial challenges. One of them is that independently trained teachers can drift arbitrarily in parameter space across task indices, making it difficult for a single $\kappa$ to fit all of them simultaneously. We address this through cross-index regularization during teacher training. In summary, our contributions are:

- **Algorithm.** We propose DIBS, a decoupled behavioral cloning approach to inductive generalization that replaces GenRL's unstable multi-task RL loop with stable supervised learning (Section 4).
- **Techniques.** We introduce cross-index regularization and confidence-based dataset filtering to address the challenges that decoupling introduces.
- **Empirics.** We conduct our evaluation on challenging continuous environments spanning increasing dynamical complexity, from simple planar motion (CAR2D) to higher-dimensional 3D navigation and attitude control (SIMPLEDRONE, DRONEATTITUDE), and a manipulation-style benchmark (REACHER). DIBS achieves 3.82× better training scalability and 6.19× better zero-shot generalization over GenRL. We also compare DIBS against four other RL and meta-RL baselines; DIBS achieves 3.02× and 3.04× better training scalability and zero-shot generalization (resp.) against the closest of these baselines (Section 5).

**Related Work.** Prior work tackles generalization in reinforcement learning through several complementary ideas. Meta-learning [43, 12, 16] prepares an agent for rapid adaptation by learning an initialization or latent task representation that can be updated with a small amount of new experience; unlike GenRL and DIBS, meta-learning approaches do not perform zero-shot generalization. Goal-conditioned RL [31, 27, 23] trains a single policy conditioned on a desired outcome, enabling zero-shot generalization by changing the goal input. Multi-task RL [30, 28, 35, 11] learns shared

representations or a shared policy across a fixed training distribution, encouraging skill reuse. Programmatic and logic-based policy representations [4, 37, 42, 6] impose symbolic structure on the policy class, improving systematic reuse and interpretability relative to purely neural policies. While all these lines of work enable generalization by adapting to a new task, conditioning on a goal, or sharing representations across a fixed training set, none directly addresses the setting we study: an indexed inductive family where task instances vary systematically with the index, and the objective is to learn a single rule that predicts how the policy should change as the index changes. GenRL [32] is, to our knowledge, the only prior method that explicitly targets this form of inductive generalization. The use of temporal logics in RL has enabled long-horizon learning across a range of settings [10, 1, 5, 9, 14, 15, 40, 38, 22, 19, 2, 39, 24, 34, 13, 25, 3]; temporal logics are useful for inductive generalization as they provide structured syntax to capture task similarities.

## 2 Preliminaries

**Markov Decision Process (MDP).** The environment in RL is modeled as a *Markov Decision Process* (MDP) $\mathcal{M} = (S, A, P, \eta)$, where $S \subseteq \mathbb{R}^n$ is a continuous state space, $A \subseteq \mathbb{R}^m$ is a continuous action space, $P(s, a, s') = p(s' \mid s, a) \in \mathbb{R}_{\geq 0}$ is the transition density of moving to $s'$ after taking action $a$ in state $s$, and $\eta : S \to \mathbb{R}_{\geq 0}$ is the initial-state distribution.

A *rollout* $\zeta \in \mathcal{Z}$ is either an infinite sequence $\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ or a finite sequence $\zeta = s_0 \xrightarrow{a_0} \cdots \xrightarrow{a_{t-1}} s_t$, where $s_i \in S$ and $a_i \in A$. We let $\mathcal{Z}_f$ denote the set of finite rollouts. A (deterministic) *policy* $\pi : \mathcal{Z}_f \to A$ maps a finite rollout to an action.

In RL, the transition probabilities are assumed to be unknown, and the MDP is accessed only through sampling. Given a policy $\pi$, a rollout is generated by sampling an initial state $s_0 \sim \eta(\cdot)$ and then iterating: take $a_i = \pi(\zeta_{0:i})$ and sample the next state $s_{i+1} \sim p(\cdot \mid s_i, a_i)$.

**Specification Language.** We specify long-horizon tasks using the temporal logic language SPECTRL [17]. A SPECTRL specification is built from a set of *atomic predicates* $\mathcal{P}_0$ over environment states. Each predicate $p \in \mathcal{P}_0$ is associated with a Boolean-valued semantics $[\![p]\!] : S \to \mathbb{B} = \{\texttt{true}, \texttt{false}\}$. We write $s \models p$ to mean $[\![p]\!](s) = \texttt{true}$. Using these predicates, the syntax of SPECTRL is $\phi ::= \texttt{achieve}\ b \mid \phi_1\ \texttt{ensuring}\ b \mid \phi_1; \phi_2 \mid \phi_1\ \texttt{or}\ \phi_2$.

Every specification $\phi$ induces a satisfaction function $[\![\phi]\!] : \mathcal{Z} \to \mathbb{B}$; we write $\zeta \models \phi$ iff $[\![\phi]\!](\zeta) = \texttt{true}$. '`achieve` ' encodes a reachability objective and ' `ensuring` ' encodes a safety constraint, while ';' and '`or`' denote sequencing and disjunction, respectively. The formal semantics is in Appendix C.

The probability with which a policy $\pi$ satisfies a specification $\varphi$ in an MDP $\mathcal{M}$ (with initial state distribution $\eta$) is given by $\Pr\left[\pi \models \mathcal{M}, \varphi\right] \triangleq \mathbb{E}_{\zeta \sim \pi, \eta}\left[\zeta \models \varphi\right]$ where $\zeta$ is a rollout of policy $\pi$ in $\mathcal{M}$ with initial state distribution $\eta$.

## 3 Inductive Generalization and Problem Statement

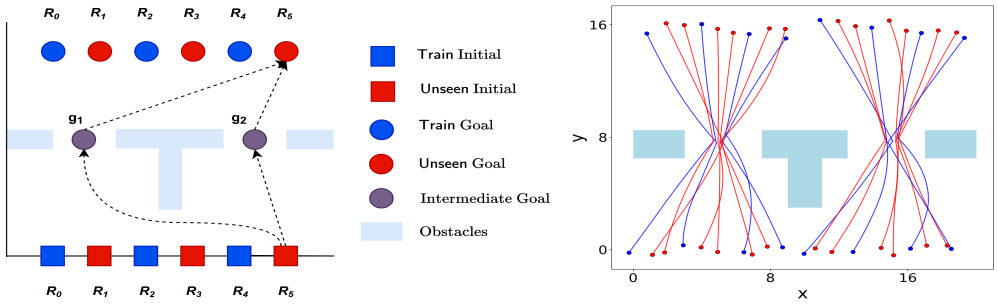
### 3.1 Inductive Task Families and Generalization

An *inductive task family* is an indexed collection $\mathcal{R} = \{\mathsf{R}_i\}_{i=0}^{L}$ of tasks where each task $\mathsf{R}_i$ pairs an MDP $\mathcal{M}_i$ with a SPECTRL specification $\varphi_i$. The family is *inductive* if all tasks share the same SPECTRL syntactic structure and differ only in the instantiation of atomic predicates or the initial state distribution. Formally, there exists a fixed specification form $\varphi(\cdot)$ and predicate instantiations $\{b_i\}_{i=0}^{L}$ such that $\varphi_i = \varphi(b_i)$, where predicate parameters evolve with the index via structured update operators (e.g., translation, scaling, or parameter drift).

Fig. 2.a illustrates an inductive task family in a 2D plane. In each task instance $\mathsf{R}_k$, the agent starts from $\mathsf{Init}_k$ and must reach $\mathsf{goal}_k$ after visiting either $\mathsf{g}_1$ or $\mathsf{g}_2$, while avoiding obstacles:

$$\varphi_k \triangleq \Big(\texttt{achieve reach}\ (\mathsf{g}_1)\ \texttt{or achieve reach}\ (\mathsf{g}_2);\ \texttt{achieve reach}\ (\mathsf{goal}_k)\Big)\ \texttt{ensuring}\ \mathcal{S}.$$

Here $\mathsf{g}_1$, $\mathsf{g}_2$ are fixed; only $\mathsf{Init}_k$ and $\mathsf{goal}_k$ shift right with $k$. The family is inductive because the same specification template is reused and only its grounding evolves.




(a) (b)

Figure 2: **Choice Benchmark in Cartesian 2D Plane. (a)** Illustration of the Choice (1-level): from initial region, reach either $g_1$ or $g_2$ then reach the final goal; initial region and the goal shift across indices. **(b)** Trajectories produced by DIBS: blue denotes Train tasks and red denotes Unseen tasks.

Given an inductive task family $\mathcal{R} = \{\mathsf{R}_i\}_{i=0}^{L}$ and training indices $\mathsf{Train} \subseteq \{0, \ldots, L\}$ with $0, L \in \mathsf{Train}$, the goal of *inductive generalization* is to produce *zero-shot* policies for unseen indices $\mathsf{Unseen} = \{0, \ldots, L\} \setminus \mathsf{Train}$, i.e., policies satisfying their tasks with high probability without further training: $\Pr[\pi_j \models \mathsf{R}_j]$ is high for many $j \in \mathsf{Unseen}$. In Fig. 2, blue regions are training indices and red regions are unseen; the algorithm trains on blue tasks only and must generalize to both.

### 3.2 Problem Statement: $\kappa$-Learning

The key insight of GenRL is that inductively related tasks admit inductively related policies. If $\pi_i$ denotes the policy for task $\mathsf{R}_i$, one can posit an inductive relation $\Omega : \Pi \to \Pi$ such that $\pi_{i+1} = \Omega(\pi_i)$. Since learning $\Omega$ directly is intractable, GenRL approximates it as an $m$-degree polynomial:

$$\Omega(\pi_i) = \kappa_m \odot [\pi_i]^m + \cdots + \kappa_1 \odot [\pi_i] + \kappa_0,$$

where $\kappa_0, \ldots, \kappa_m$ share the dimension of the neural policy parameters, $\odot$ is the Hadamard product, and $[\pi_i]^r$ denotes elementwise power. Inductive generalization thus reduces to learning these $\kappa$-coefficients from Train and unrolling the learned template from $\pi_0$ to generate zero-shot policies.

A subtlety arises because policies need not evolve smoothly across all indices even when tasks do. Consider Fig. 2 where policies on the far left route through $g_1$ and those on the far right through $g_2$, with a discontinuous switch in between. Following GenRL, we handle this via a *compositional* reduction: complex SPECTRL specifications are decomposed into *reach-avoid* subspecifications of the form $\varphi(\mathcal{G}, \mathcal{S}) := \big(\texttt{achieve } \mathcal{G}\big)\ \texttt{ensuring } \mathcal{S}$, $\kappa$-learning is applied to each, and the results are composed. This ensures inductive assumptions are placed only on the simplest specifications SPECTRL can encode; see [32, 18] for details. The central learning problem is:

**Definition 3.1 ($\kappa$-learning)** *Given an inductive family of reach-avoid tasks $\mathcal{R} = \{\mathsf{R}_i\}_{i=0}^{L}$, training indices $\mathsf{Train} = \{i_1 < \cdots < i_n\}$ with $i_1 = 0$, $i_n = L$, polynomial degree $m$, and base policy $\pi_0$, learn coefficients $\kappa_0, \ldots, \kappa_m$ such that*

$$\kappa_0^*, \cdots, \kappa_m^* \in \underset{\kappa_0, \cdots, \kappa_m}{\arg\max} \sum_{i \in \mathsf{Train}} \Pr[\pi_i \models \mathsf{R}_i],$$

*where $\pi_i$ is obtained by unrolling the $\kappa$-polynomial from $\pi_0$.*

**GenRL's approach and its limitation.** GenRL solves Def. 3.1 via a coupled RL loop (Fig. 1.c): it unrolls the current $\kappa$ to generate $\{\pi_i\}_{i \in \mathsf{Train}}$, executes these policies on their training tasks, and updates $\kappa$ from the aggregated multi-task reward. The fundamental limitation is that $\kappa$-updates depend on policies that are themselves still learning — errors compound across the loop, and instability worsens as $|\mathsf{Train}|$ grows. DIBS resolves this by decoupling the two phases, as described in Section 4.

## 4 Algorithm

Algorithm 1 (Appendix E) describes our *decoupled* training procedure, DIBS, which learns task-specific teacher policies $\hat{\pi}_i$ for their respective tasks $\mathsf{R}_i$ in Stage I, and then uses these teachers

as oracles to learn a *general* policy template in Stage II. Specifically, Stage II learns the template parameters $\kappa$ such that each task-specific policy instance obtained by unrolling the $\kappa$-template imitates the corresponding teacher.

### 4.1 Stage I: Teacher-labeled dataset aggregation

Stage 1 constructs, for each training index $i \in \mathsf{Train}$, a final dataset $\widetilde{\mathcal{D}}_i$ of teacher-labeled state–action pairs. This can be accomplished simply by learning teacher policies $\pi_i$ for all $i \in \mathsf{Train}$ using a standard RL approach independently. This simple pipeline faces two critical issues:

- **(C1) Cross-index teacher drift.** A task may be successfully accomplished by a diverse set of policies. As Stage II must fit all of these teacher policies using a *single* set of template coefficients $\kappa$, which may not be possible if the teachers are trained independently in an unconstrained manner—as adjacent teachers $\hat{\pi}_i$ and $\hat{\pi}_{i^-}$ can end up being very different, even when $\mathsf{R}_i$ and $\mathsf{R}_{i^-}$ are similar, making it hard for a single $\kappa$ to fit all indices simultaneously. We address this by adding *cross-index regularization* during teacher training (see Section 4.1.1).
- **(C2) Coverage and unreliable labels.** While these test-time rollouts mostly visit *high-quality states*, i.e., the subset of the state space where the teacher has been sufficiently trained and consistently achieves high specification satisfaction, they provide limited coverage of the set of states on which a template-generated policy may need to act during unrolling. Hence, the candidate state set must be *expanded* to include a wider range of states. However, diversifying the sampling procedure also runs the risk of witnessing *low-quality states* that lie far outside the region that the teacher has explored, and hence, the teacher actions can be unreliable in such states. We address this by *training a confidence filter* (Section 4.1.2).

#### 4.1.1 Learning teacher policies with cross-index regularization.

For each $\mathsf{R}_i$ with $i \in \mathsf{Train}$, we train a teacher policy $\hat{\pi}_i$ s.t., (a) it achieves high reach-avoid success w.r.t its task $\mathsf{R}_i$, and (b) the policy $\hat{\pi}_i$ stays close to the teacher policy learned at the previous training index. The teacher $\hat{\pi}_i$ for task $\mathsf{R}_i$ is trained by minimizing

$$\mathcal{L}^{(i)}_{\text{teacher}} = \mathcal{L}^{(i)}_{\text{RL}} + \lambda_{\text{x}}\, \mathcal{L}^{(i)}_{\text{xreg}}, \qquad\qquad \mathcal{L}^{(i)}_{\text{xreg}} \triangleq \|\hat{\pi}_i - \hat{\pi}_{i^-}\|_2^2,$$

where $\mathcal{L}^{(i)}_{\text{RL}}$ is the single-task RL objective and $\mathcal{L}^{(i)}_{\text{xreg}}$ is the *cross-index regularization* on reach-avoid task $\mathsf{R}_i$; $\lambda_{\text{x}} \geq 0$ controls the strength of the cross-index regularization.

The *cross-index regularization* is given by the squared Euclidean distance between the teacher parameter vectors, where $i^-$ denotes the previous element of $\mathsf{Train}$ before $i$ (so $i^- < i$ and there is no training index between them) and $\|\cdot\|_2$ is the Euclidean norm over policy parameter vectors (i.e., we treat $\hat{\pi}_i$ and $\hat{\pi}_{i^-}$ as vectors of network weights). This term provides a soft continuity constraint across indices such that it biases the optimizer toward solutions for $\mathsf{R}_i$ that can be obtained by a small change from the previous teacher, while still allowing the policy to deviate when the task requires it.

The teacher policy for this loss function can be trained using a standard RL algorithm (Note: For the base policy $\pi_0$, $\mathcal{L}^{(0)}_{\text{xreg}} = 0$ as there is no previous task).

#### 4.1.2 Confidence-filtered state accumulation.

After training $\hat{\pi}_i$, we build a preliminary dataset $\mathcal{D}_i$ by sampling states from the teacher's on-policy distribution $d^{\hat{\pi}_i}$. At the same time, to combat the *coverage* problem (see challenge C2 above), we diversify our samples by also sampling from two other sources: the replay-buffer distribution $\mathrm{Replay}_i$, and the reset distribution $\mathrm{Reset}_i$. Here, $d^{\hat{\pi}_i}$ collects states visited in specification-satisfying *on-policy* rollouts of the *converged* teacher $\hat{\pi}_i$ during test-time (i.e., after training has converged). $\mathrm{Replay}_i$ samples states from the teacher's replay buffer accumulated over training (including early exploration states and other perturbation-induced states), and $\mathrm{Reset}_i$ samples initial states produced by environment resets or randomized initializations.

However, as discussed in the challenge C2 above, the data points from the replay/reset buffers may be of *low quality* as the teacher may not have witnessed these samples during its exploration. Hence, to avoid learning our $\kappa$ on a noisy dataset, we train a *confidence filter* that assigns a confidence probability $p_i^{\text{conf}}(s)$ to states $s$, indicating whether $s$ is a high- or a low-quality state.

**Confidence filter training.** To train the confidence filter, we maintain two state buffers: (i) a high-quality state buffer $\mathcal{B}_i^{\text{high}}$ containing states visited during teacher training in rollouts that satisfy the specification for task $i$, and (ii) a low-quality state buffer $\mathcal{B}_i^{\text{low}}$ containing states from early exploration, perturbed observations, and reset-induced states, and, more generally, states from rollouts that do not satisfy (or have lower satisfaction of) the specification. We assign labels $y = 1$ for $s \sim \mathcal{B}_i^{\text{high}}$ and $y = 0$ for $s \sim \mathcal{B}_i^{\text{low}}$. The confidence filter is trained with binary cross-entropy:

$$\mathcal{L}_{\text{conf}}^{(i)} = \mathbb{E}_{s\sim\mathcal{B}_i^{\text{high}}}\big[-\log(p_i^{\text{conf}}(s))\big] \quad + \quad \mathbb{E}_{s\sim\mathcal{B}_i^{\text{low}}}\big[-\log\big(1 - p_i^{\text{conf}}(s)\big)\big]\,.$$

**Aggregating the samples.** Once $p_i^{\text{conf}}(\cdot)$ is obtained, we convert the preliminary dataset $\mathcal{D}_i$ into a labeled dataset $\widetilde{\mathcal{D}}_i$ by retaining only those candidates that the confidence filter deems high-quality. Specifically, we keep a candidate state $s \in \mathcal{D}_i$ only if $p_i^{\text{conf}}(s) \geq \tau_c$, where $\tau_c \in (0, 1)$ is a fixed threshold. For each retained state, we query the teacher for an action label $a = \hat{\pi}_i(s)$ and add $(s, a)$ to the final dataset: $\widetilde{\mathcal{D}}_i = \{(s, a) : s \in \mathcal{D}_i,\ p_i^{\text{conf}}(s) \geq \tau_c,\ a = \hat{\pi}_i(s)\}$. This retains the coverage benefits of replay/reset while discarding candidates likely to yield unreliable labels.

**Implementation Details.** Instead of training the teacher policy and the confidence filter separately, we jointly optimize both via $\mathcal{L}_{\text{teacher}-\text{conf}}^{(i)} = \mathcal{L}_{\text{teacher}}^{(i)} + \lambda_{\text{conf}}\,\mathcal{L}_{\text{conf}}^{(i)}$, where $\lambda_{\text{conf}} \in [0, 1]$ controls how strongly we train the confidence filter. We train a single neural network for $\mathcal{L}_{\text{teacher}-\text{conf}}^{(i)}$ with a shared trunk $h_i$ and two heads: an action (teacher policy) head and a confidence-filter head. Therefore, for a state $s$, $z = h_i(s), \quad a = \text{ActHead}_i(z), \quad c_i(s) = \text{ConfHead}_i(z)$. Here $a$ is the action output by the teacher policy at state $s$, and $c_i(s) \in \mathbb{R}$ is a confidence value indicating whether $s$ is a high-quality state. We convert this confidence value to a probability via $p_i^{\text{conf}}(s) \triangleq \text{Sigmoid}(c_i(s))$. Thus, larger $p_i^{\text{conf}}(s)$ indicates higher confidence that $s$ is a high-quality state.

### 4.2 Stage II: Learning $\kappa$ coefficients via Behavioral Cloning

We learn the template coefficients $\kappa$ by fitting the final datasets $\bigcup_{i\in\mathsf{Train}} \widetilde{\mathcal{D}}_i$. We fit $\kappa$ using *behavioral cloning (BC)*, a standard imitation-learning objective that trains a policy to match the demonstrated action on a dataset of state–action pairs. In our case, the demonstrations come from the dataset accumulated in Stage I: for each training index $i \in \mathsf{Train}$, we construct a final dataset $\widetilde{\mathcal{D}}_i$ of state–action pairs, where each action is labeled by the teacher policy $\hat{\pi}_i$. Unlike standalone BC, which fits a single policy to a single dataset, we use BC to fit a *shared* $\kappa$-template whose unrolling generates a policy for each training index, using all $\{\widetilde{\mathcal{D}}_i\}_{i\in\mathsf{Train}}$ as supervision. BC fits $\kappa$ by minimizing the discrepancy between the template-generated actions (from $\pi_i$) and the teacher actions on the teacher-labeled datasets. Using the squared-error behavioral cloning loss, we minimize

$$\mathcal{L}_{\text{BC}}(\kappa) = \sum_{i\in\mathsf{Train}} \frac{1}{|\widetilde{\mathcal{D}}_i|} \sum_{(s,a)\in\widetilde{\mathcal{D}}_i} \|\pi_i(s) - a\|_2^2\,.$$

Here, $\pi_i$ is the policy for task $\mathsf{R}_i$ obtained by unrolling the $\kappa$-polynomial. For unrolling formalism, see Appendix D.1. We update $\kappa$ with a first-order optimizer:

$$\kappa \leftarrow \kappa - \eta\,\nabla_\kappa \mathcal{L}_{\text{BC}}(\kappa),$$

alternating between (i) unrolling the current $\kappa$ to obtain the template-generated policies $\{\pi_i\}_{i\in\mathsf{Train}}$ and (ii) taking gradient steps on $\mathcal{L}_{\text{BC}}$ until the loss plateaus. The output is a set of learned template coefficients $\kappa$ that can be unrolled from base policy $\pi_0$ to produce policies for any index in the family.

## 5 Empirical Evaluation

Our empirical analysis is designed to evaluate three metrics: (a) training scalability, (b) zero-shot generalization, and (c) the correlation between training scalability and zero-shot generalization.

### 5.1 Experimental Setup

**Baselines.** We compare our method DIBS to five other methods. **ARS** [26] trains a single policy using aggregated rollouts across all training indices in Train, with no explicit index-evolution template.

**BC** [36, 21, 29] is a standalone imitation learning approach that trains a single policy on the pooled teacher-labeled dataset $\bigcup_{i \in \mathsf{Train}} \widetilde{\mathcal{D}}_i$, with no explicit template. While neither ARS nor BC is a generalization method, we also compare with **MAML** [12] and **VariBAD** [43], which are strong meta-RL generalization baselines. Finally, GenRL is an inductive generalization approach that learns the $\kappa$-coefficients using an ARS-style RL loop. DIBS borrows the high-level architecture of GenRL but uses the procedure in Section 4 to train $\kappa$-coefficients.

**Environments.** We evaluate on four continuous-control environments: CAR2D, REACHER, SIMPLEDRONE, and DRONEATTITUDE. CAR2D is a planar car-like navigation model. The state is the agent position $s = (x, y) \in \mathbb{R}^2$, and the action is $a = (v_f, \theta) \in \mathbb{R}^2$, where $v_f$ denotes the forward speed and $\theta$ denotes the heading direction. REACHER is a two-link planar arm where the state is the end-effector position $s = (x, y) \in \mathbb{R}^2$ and the action specifies the two joint angles $a = (\theta_1, \theta_2) \in \mathbb{R}^2$ (Fig. 1.a). SIMPLEDRONE models holonomic 3D navigation with state $s = (x, y, z) \in \mathbb{R}^3$ and continuous velocity actions $a = (v_x, v_y, v_z) \in \mathbb{R}^3$. DRONEATTITUDE augments 3D position with attitude variables, using $s = (x, y, z, \psi, \theta) \in \mathbb{R}^5$, and controls the agent through body-frame velocity commands and angular rates, $a = (v_f, v_s, v_u, \omega_\psi, \omega_\theta) \in \mathbb{R}^5$. See Figure 5 in Appendix for illustrations.

Overall, these environments span increasing dynamical complexity, from simple planar motion (CAR2D) to higher-dimensional 3D navigation and attitude control (SIMPLEDRONE, DRONEATTITUDE), and a manipulation-style benchmark (REACHER).

**Inductive Task Families.** We create several inductive task families over the environments (see Appendix G). For REACHER, the inductive task families consist of variations of the tower-destacking problem (Fig. 4) with the tower varying in height from 8 to 10 blocks. For the other environments, we use $k$-REACHABILITY (long-horizon sequencing) where the initial state distribution and all the $k$ intermediate points shift to the right as the task index increases (Appendix Fig. 6.a). In CAR2D we also consider CHOICE (branching via disjunction) with 1-level (Fig. 2) and 2-level branching variants (Appendix Fig. 7.b).

**Train/unseen splits and index spacing.** Each benchmark defines task instances indexed by $\{0, \ldots, L\}$. A training run chooses an ordered subset of training tasks $\mathsf{Train} = \{i_1 < \cdots < i_n\}$ with $i_1 = 0$ and $i_n = L$, and the remaining unseen tasks $\mathsf{Unseen} = \{0, \ldots, L\} \setminus \mathsf{Train}$.

**Metrics.** Given a policy and a specification (a single task), we estimate the probability of specification satisfaction via repeated rollouts of the policy. A policy is said to *succeed* if this estimate exceeds a given threshold value $\delta$. For all our experiments, we use $1000$ rollouts and $\delta = 0.9$.

For a generalization baseline and an inductive task family, we define two metrics: (a) successful training ratio, and (b) zero-shot generalization ratio. Given a training set Train, the *successful training ratio* is given by $\frac{|i \in \mathsf{Train} : \pi_i \text{ succeeds}|}{|\mathsf{Train}|}$, where $\pi_i$ is the policy obtained by the baseline for the training task indexed by $i$. In our training plots, we report the successful training ratio on the $y$-axis against the number of training tasks $|\mathsf{Train}|$ on the $x$-axis. Given a testing set Unseen, the *absolute zero-shot generalization* is given by $|i \in \mathsf{Unseen} : \pi_i \text{ succeeds}|$. The *zero-shot generalization ratio* is given by $\frac{|i \in \mathsf{Unseen} : \pi_i \text{ succeeds}|}{|\mathsf{Unseen}|}$, i.e., the fraction of unseen tasks solved. In our zero-shot generalization plots, we report this ratio on the $y$-axis against the number of training tasks $|\mathsf{Train}|$ on the $x$-axis.

### 5.2 Results and Observations

Fig. 3 summarizes results for some $k$-REACHABILITY tasks across multiple inductive task families: CAR2D, SIMPLEDRONE, DRONEATTITUDE; plots for remaining benchmarks are in the Appendix H.

**Training scalability.** Training scalability is evaluated as successful training ratio vs. $|\mathsf{Train}|$, as demonstrated in Fig. 3a. Higher values indicate better scalability. Fig. 3a shows that our approach DIBS maintains the strongest training scalability. Aggregated across all benchmarks and training configurations, DIBS improves the fraction of training indices solved by $3.82\times$ relative to the other inductive generalization approach GenRL. We attribute this success to our decoupled approach that reduces $\kappa$-learning to a simple supervised learning objective (in the form of behavioral cloning), as opposed to a complex multi-reward aggregation RL-style loop in GenRL.

**Zero-Shot Generalization.** DIBS also achieves the strongest zero-shot generalization across benchmarks, both in the ratio (Fig. 3b) and in the absolute value (Fig. 3c). Because template fitting

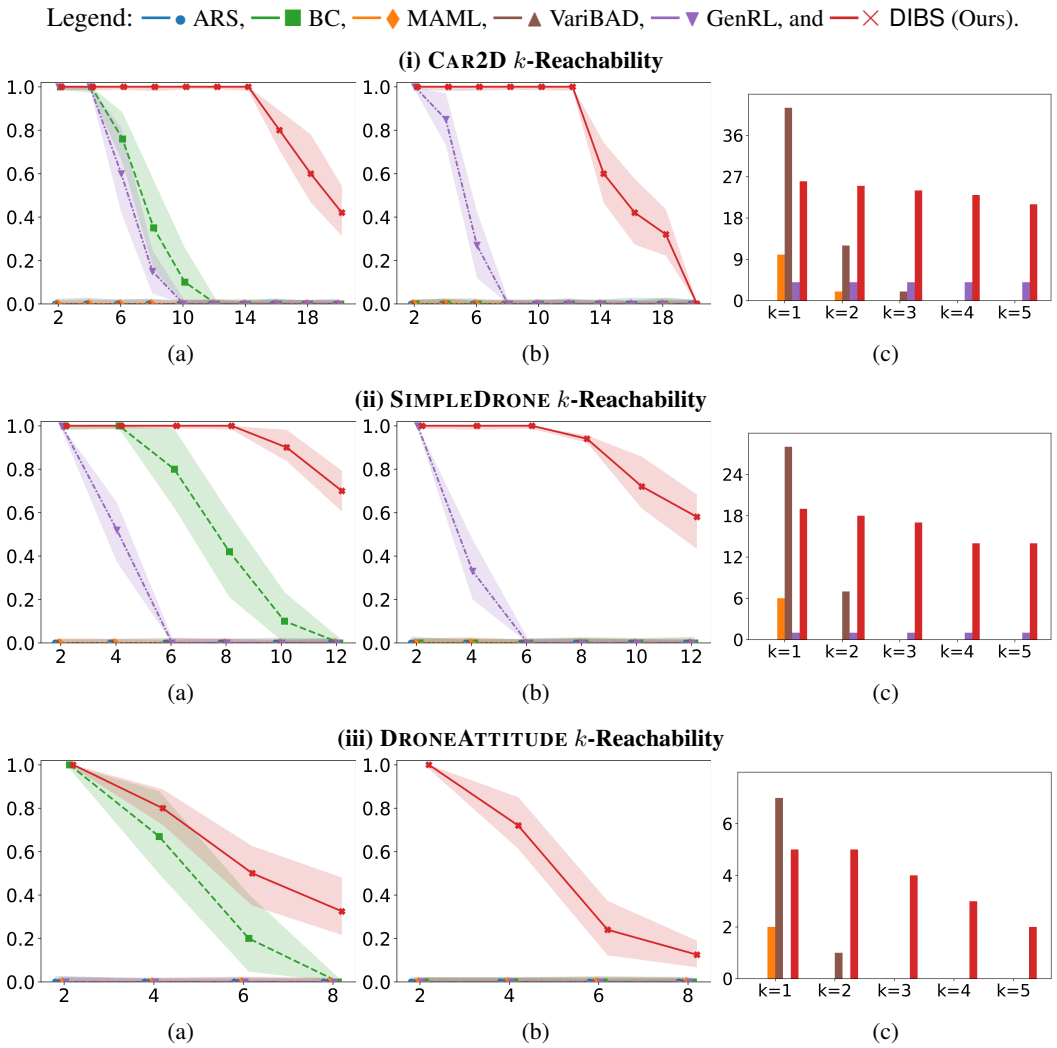


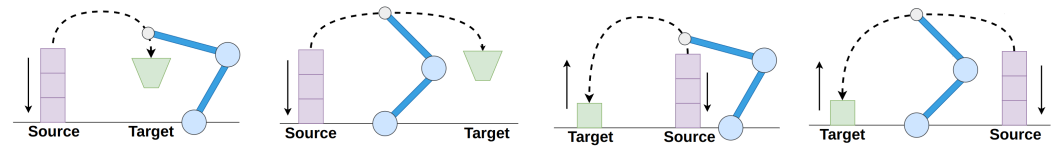

Figure 3: **Scalability plots**. Each row corresponds to one environment on $k$-reachability tasks. **Column (a)** shows the successful training ratio ($y$-axis) versus number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 5$. **Column (b)** shows the zero-shot generalization ratio ($y$-axis) versus number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 5$. **Column (c)** shows absolute zero-shot generalization ($y$-axis) for $k \in \{1, \ldots, 5\}$ ($x$-axis). We report mean ($\pm$ standard deviation) performance across **ten** random seeds.

remains stable across a wider range of training configurations, the learned $\kappa$ can be unrolled to solve substantially more unseen indices. Aggregated across all benchmarks and training configurations, DIBS improves the fraction of unseen indices solved by $6.19\times$ relative to GenRL.

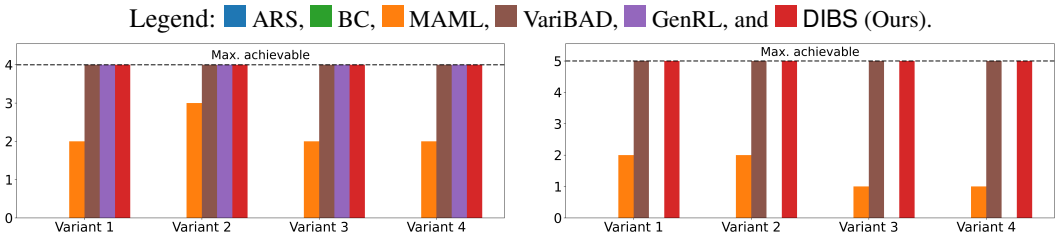
Notably, **BC**, which is behavioral cloning without the template scales well in $|\mathsf{Train}|$ (Fig. 3a); however, it does not generalize well (Fig. 3b–c). This is not unexpected, since in the absence of a template **BC** has no context to generalize beyond its supervised training data. This also demonstrates the strength of the template-based generalization approach pursued in inductive generalization.

**Correlation between training scalability and zero-shot generalization.** We observe a consistent and strong correlation between training success and generalization, i.e., when a method fails to solve its training indices at larger $|\mathsf{Train}|$, its zero-shot generalization typically degrades as well (Fig. 3a–b). Though this is an observed trend rather than a causal claim, it suggests that scalable training dynamics are closely tied to extracting generalizable structure from larger training index sets.

MAML and VariBAD can perform competitively in simpler settings, but they do not scale reliably to long-horizon or branching tasks. As the task horizon increases, for example, for larger $k$ in the $k$-REACHABILITY experiments in Fig. 3c, their training reliability and zero-shot generalization




(a) Pick-and-drop (same side) — Variant 1 (b) Pick-and-drop (opposite side) — Variant 2 (c) Pick-and-stack (same side) — Variant 3 (d) Pick-and-stack (opposite side) — Variant 4




(e) 8 blocks (f) 10 blocks

Figure 4: **REACHER task variants and zero-shot generalization results.** Top row: specification illustrations for the four REACHER tower pick-and-place variants. Bottom row: absolute zero-shot generalization results for the 8-block and 10-block tower settings. The $y$-axis in the bottom-row plots shows absolute zero-shot generalization, and the $x$-axis denotes the four Reacher variants shown in the top row.

degrade relative to the template-based methods. A similar pattern appears in the CHOICE benchmarks, where branching decisions are required and both methods struggle because they do not model the branching structure; see Appendix Fig. 13. These results show that while MAML and VariBAD can work in easier regimes, their performance deteriorates as the task structure becomes longer-horizon or more compositional. This trend is consistent with the behavior observed for GenRL, although GenRL benefits from an explicit template structure.

**Ablation Study: Decoupling teacher learning from $\kappa$-learning.** Since the $\kappa$ optimization is embedded within this ARS loop, GenRL cannot easily benefit from stronger single-task RL methods. Consequently, when ARS degrades, GenRL's template learning also breaks down (Fig. 4). For example, REACHER becomes harder as the number of displaced blocks grows due to longer horizons and tighter manipulation. In such cases, ARS frequently fails to learn a strong controller, which impacts GenRL's $\kappa$-template learning due to poor reward signals and unstable $\kappa$ updates.

In contrast, DIBS separates teacher learning from template fitting. We train each per-index teacher $\hat{\pi}_i$ with any suitable RL algorithm (e.g., PPO), then fit the same template class via behavioral cloning on teacher-labeled datasets. Because template optimization is independent of the teacher RL backbone, DIBS is capable of using stronger teachers and even domain-specific teachers out of the box, which directly translates into better supervision, thereby enabling stable learning even in harder settings. Empirically, this produces a usable template where GenRL fails (Fig. 4), and the same trend appears in SIMPLEDRONE and DRONEATTITUDE (Fig. 3.a–3.c).

**Ablation Study: Cross-index drift regularization.** We ablate the cross-index drift regularizer used in Stage I teacher training on SIMPLEDRONE. Removing the regularizer ($\lambda_x = 0$) causes a sharp drop in zero-shot generalization: the best run decreases from 26 solved unseen tasks (with regularization) to 3 solved unseen tasks (without regularization). This highlights that cross-index drift regularization is important for producing a teacher sequence that can be fit smoothly by a single template, which in turn improves generalization to unseen task indices.

## 6 Conclusion

We introduce a novel inductive generalization approach to RL that, by decoupling teacher policy learning from template training, (a) addresses a fundamental instability in prior work and (b) eliminates dependence on a specific RL algorithm, both of which limit scalability to larger task families. Experiments on complex temporal logic benchmarks confirm that our approach significantly outperforms existing methods on both training stability and zero-shot generalization.

**Acknowledgment**

Research reported in this publication was partly supported by an Amazon Research Award, Fall 2024

## References


[1] Aksaray, D., Jones, A., Kong, Z., Schwager, M., and Belta, C. Q-learning for robust satisfaction of signal temporal logic specifications. In *Conference on Decision and Control (CDC)*, pp. 6565–6570. IEEE, 2016.

[2] Alur, R., Bansal, S., Bastani, O., and Jothimurugan, K. A framework for transforming specifications in reinforcement learning. In *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, pp. 604–624. Springer, 2022.

[3] Alur, R., Bansal, S., Bastani, O., and Jothimurugan, K. Specification-guided reinforcement learning. *Communications of the ACM*, 69(2):80–87, 2026.

[4] Bastani, O., Pu, Y., and Solar-Lezama, A. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018.

[5] Brafman, R., De Giacomo, G., and Patrizi, F. LTLf/LDLf non-markovian rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[6] Cao, Y., Li, Z., Yang, T., Zhang, H., Zheng, Y., Li, Y., Hao, J., and Liu, Y. GALOIS: boosting deep reinforcement learning via generalizable logic synthesis. *Advances in Neural Information Processing Systems*, 35:19930–19943, 2022.

[7] Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. In *International conference on machine learning*, pp. 1282–1289. PMLR, 2019.

[8] Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

[9] De Giacomo, G., Iocchi, L., Favorito, M., and Patrizi, F. Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pp. 128–136, 2019.

[10] Dohmen, T., Perez, M., Somenzi, F., and Trivedi, A. Regular reinforcement learning. In Gurfinkel, A. and Ganesh, V. (eds.), *Computer Aided Verification*, pp. 184–208, Cham, 2024. Springer Nature Switzerland.

[11] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

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

[13] Guo, Z., Işık, İ., Ahmad, H., and Li, W. One subgoal at a time: Zero-shot generalization to arbitrary linear temporal logic requirements in multi-task reinforcement learning. *arXiv preprint arXiv:2508.01561*, 2025.

[14] Hasanbeig, M., Abate, A., and Kroening, D. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

[15] Hasanbeig, M., Kantaros, Y., Abate, A., Kroening, D., Pappas, G. J., and Lee, I. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *Conference on Decision and Control (CDC)*, pp. 5338–5343, 2019.

[16] Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[17] Jothimurugan, K., Alur, R., and Bastani, O. A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

[18] Jothimurugan, K., Bansal, S., Bastani, O., and Alur, R. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34:10026–10039, 2021.

[19] Jothimurugan, K., Bansal, S., Bastani, O., and Alur, R. Specification-guided learning of Nash equilibria with high social welfare. In *International Conference on Computer Aided Verification*, pp. 343–363. Springer, 2022.

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

[21] Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[22] Li, X., Vasile, C.-I., and Belta, C. Reinforcement learning with temporal logic rewards. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3834–3839. IEEE, 2017.

[23] Liu, M., Zhu, M., and Zhang, W. Goal-conditioned reinforcement learning: Problems and solutions. *arXiv preprint arXiv:2201.08299*, 2022.

[24] Liu, Z., Guo, Z., Yao, Y., Cen, Z., Yu, W., Zhang, T., and Zhao, D. Constrained decision transformer for offline safe reinforcement learning. In *International conference on machine learning*, pp. 21611–21630. PMLR, 2023.

[25] Majumdar, R., Salamati, M., and Soudjani, S. Regret-free reinforcement learning for temporal logic specifications. In *Forty-second International Conference on Machine Learning*, 2025.

[26] Mania, H., Guy, A., and Recht, B. Simple random search of static linear policies is competitive for reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

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

[28] Oh, J., Singh, S., Lee, H., and Kohli, P. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, pp. 2661–2670. PMLR, 2017.

[29] Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., and Peters, J. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[30] Sodhani, S., Zhang, A., and Pineau, J. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, pp. 9767–9779. PMLR, 2021.

[31] Sohn, S., Oh, J., and Lee, H. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. *Advances in neural information processing systems*, 31, 2018.

[32] Subramanian, V., Kushwah, R., Roy, S., and Bansal, S. Inductive generalization in reinforcement learning from specifications. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 277–298. Springer, 2025.

[33] Sutton, R. S., Barto, A. G., et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[34] Svoboda, J., Bansal, S., and Chatterjee, K. Reinforcement learning from reachability specifications: Pac guarantees with expected conditional distance. In *Forty-first International Conference on Machine Learning*, 2024.

[35] Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[36] Torabi, F., Warnell, G., and Stone, P. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[37] Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pp. 5045–5054. PMLR, 2018.

[38] Xu, Z. and Topcu, U. Transfer of temporal logic formulas in reinforcement learning. In *International Joint Conference on Artificial Intelligence*, pp. 4010–4018, 7 2019.

[39] Yang, C., Littman, M., and Carbin, M. On the (in) tractability of reinforcement learning for ltl objectives. *arXiv preprint arXiv:2111.12679*, 2021.

[40] Yuan, L. Z., Hasanbeig, M., Abate, A., and Kroening, D. Modular deep reinforcement learning with temporal logic specifications. *arXiv preprint arXiv:1909.11591*, 2019.

[41] Zhang, C., Vinyals, O., Munos, R., and Bengio, S. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

[42] Zhu, H., Xiong, Z., Magill, S., and Jagannathan, S. An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, pp. 686–701, 2019.

[43] Zintgraf, L., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K., and Whiteson, S. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. In *International Conference on Learning Representations*, 2019.

# Appendix

## A Limitations

Our approach improves inductive generalization by learning a shared policy-evolution template, but it comes with a few limitations. First, the method still depends on having sufficiently strong per-index teachers; if a teacher fails on a hard index, the supervised signal used to fit the template can degrade. Second, as the number of training indices grows, the aggregated supervision pool grows and the template (and the policies it generates) may require higher-capacity networks to avoid underfitting, which increases optimization difficulty and can introduce a tradeoff between expressiveness and smooth template fitting. In addition, the added design components (cross-index regularization, candidate-state mixture, confidence threshold) introduce hyperparameters that may require tuning across environments.

## B Societal Impact

This work can help make real-world robotics more efficient by reducing how often systems must be retrained from scratch when tasks change in a structured way, such as shifting goals, initial conditions, or obstacle layouts. By learning a reusable policy-evolution rule from a small set of trained task policies, the approach can cut repeated engineering effort and enable faster deployment across related task variants, which is practical for domains like navigation, manipulation, and autonomy. We also explicitly account for safety through reach-avoid task formulations, where success is defined jointly by reaching the goal and maintaining safety constraints. That said, any deployment in physical systems still requires careful validation under distribution shift and conservative safety checks, since real environments can differ from the assumed task family.

## C SPECTRL Specification Language

Let a subrollout of $\zeta$ be the subsequence $\zeta_{\ell:k} = s_\ell \xrightarrow{a_\ell} \cdots \xrightarrow{a_{k-1}} s_k$.

For a finite rollout $\zeta$ of length $t$, satisfaction is defined as:

| | |
|---|---|
| $\zeta \models \texttt{achieve}\ b$ | if $\exists\, \tau \le t,\ s_\tau \models b$ |
| $\zeta \models \phi\ \texttt{ensuring}\ b$ | if $\zeta \models \phi$ and $\forall\, \tau \le t,\ s_\tau \models b$ |
| $\zeta \models \phi_1; \phi_2$ | if $\exists\, \tau < t,\ \zeta_{0:\tau} \models \phi_1$ and $\zeta_{\tau+1:t} \models \phi_2$ |
| $\zeta \models \phi_1\ \texttt{or}\ \phi_2$ | if $\zeta \models \phi_1$ or $\zeta \models \phi_2$. |

## D More Algorithm Details

The complete algorithm is given in Algorithm 1.

### D.1 Template unrolling

**Unrolling the $\kappa$-template across an index gap.** Given a policy at index $i$, applying the $\kappa$-parameterized update once produces a policy for the next index; applying it repeatedly produces a policy for a larger index gap. We write this procedure as

$$\texttt{UnrollPolicy}(\kappa, \pi, \Delta, m),$$

which takes an input policy $\pi$, applies the $\kappa$-parameterized update $\Delta$ times (with degree $m$), and returns the resulting policy.

**Unrolling across the training indices.** Starting from the base policy $\pi_0$ at index $i_1 = 0$, we obtain template-generated policies at the training indices by unrolling across successive gaps. Let $\mathsf{Train} = \{i_1 < \cdots < i_n\}$ with $i_1 = 0$. For each $k = 2, \ldots, n$, define the gap $\Delta_k \triangleq i_k - i_{k-1}$ and set

$$\pi_{i_k} = \texttt{UnrollPolicy}(\kappa, \pi_{i_{k-1}}, \Delta_k, m).$$

Thus, $\pi_{i_k}$ denotes the template-generated policy for the training task at index $i_k$.

**Algorithm 1** DIBS: Decoupled $\kappa$-learning via Behavioral Cloning

**Require:** Task indices $\{0, \dots, L\}$; training index set $\mathsf{Train} = \{i_1 < \cdots < i_n\}$ with $i_1 = 0$ and $i_n = L$; base policy $\pi_0$; learning rate $\eta$; aggregation budget $M$; polynomial degree $m$; teacher policies $\{\hat{\pi}_i\}_{i \in \mathsf{Train}}$; preliminary datasets $\{\mathcal{D}_i\}_{i \in \mathsf{Train}}$; confidence probability $\{p_i^{\mathrm{conf}}(\cdot)\}_{i \in \mathsf{Train}}$; threshold $\tau_c$ and index gap $\Delta$.

**Stage I: Teacher-labeled, confidence-filtered dataset creation**
▷ Assume teachers $\{\hat{\pi}_i\}$ are trained using Section 4.1, and preliminary datasets $\{\mathcal{D}_i\}$ are collected as in Section 4.1.2.

**for all** $i \in \mathsf{Train}$ **do**
  $\widetilde{\mathcal{D}}_i \leftarrow \emptyset$
  **while** $|\widetilde{\mathcal{D}}_i| < M$ **do**
    Sample $s \sim \mathcal{D}_i$
    **if** $p_i^{\mathrm{conf}}(s) \geq \tau_c$ **then** ▷ retain high-quality states
      $a \leftarrow \hat{\pi}_i(s)$
      $\widetilde{\mathcal{D}}_i \leftarrow \widetilde{\mathcal{D}}_i \cup \{(s, a)\}$

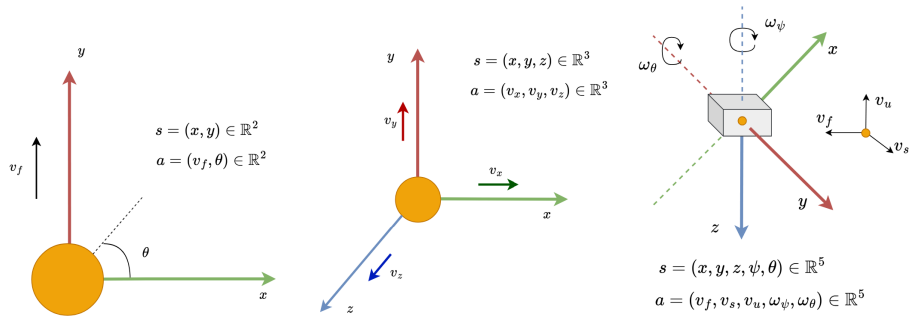
**Stage II: Fit $\kappa$ by Behavioral Cloning**
Initialize $\kappa$ (degree $m$)
**repeat**
  $\pi_{i_1} \leftarrow \pi_0$ ▷ $i_1 = 0$
  **for** $k = 2$ to $n$ **do**
    Unroll $\kappa$ from $\pi_{i_{k-1}}$, $\Delta$ times to obtain $\pi_{i_k}$
  $\mathcal{L}_{\mathrm{BC}} \leftarrow 0$
  **for** $k = 1$ to $n$ **do**
    $\mathcal{L}_{\mathrm{BC}} \leftarrow \mathcal{L}_{\mathrm{BC}} + \frac{1}{|\widetilde{\mathcal{D}}_{i_k}|} \sum_{(s,a) \in \widetilde{\mathcal{D}}_{i_k}} \|\pi_{i_k}(s) - a\|_2^2$
  $\kappa \leftarrow \kappa - \eta \nabla_\kappa \mathcal{L}_{\mathrm{BC}}$
**until** $\mathcal{L}_{\mathrm{BC}}$ is below a threshold or stagnates
**return** $\kappa$

# E Implementation Details

**Teacher training algorithms.** For each training index $i \in \mathsf{Train}$, we train a task-specific teacher policy $\hat{\pi}_i$ using standard RL on $\mathsf{R}_i$. We use ARS for CAR2D, and PPO for REACHER, SIMPLEDRONE, and DRONEATTITUDE. These choices yield stable expert-level teachers for subsequent dataset labeling. To parameterize how *densely* training indices cover the index line, we use a spacing parameter $\Delta \in \mathbb{N}$ between training indices to construct $\mathsf{Train} = \{0, \Delta, 2\Delta, \dots, L\}$ (truncated to lie in $\{0, \dots, L\}$). Larger $\Delta$ yields sparser supervision along the index family and typically makes zero-shot generalization harder because neighboring indices are less constrained by training data.

**Teacher and confidence-head architectures.** We train a single neural network per index $i$ with a shared MLP trunk $h_i$ and two heads: an action (teacher policy) head and a confidence head. For CAR2D and REACHER, the trunk is a 1-layer MLP with 8 hidden units. For the higher-dimensional benchmarks, we scale capacity while keeping the architecture class consistent: for SIMPLEDRONE we use a 2-layer MLP with 64 hidden units per layer, and for DRONEATTITUDE we use a 2-layer MLP with 128 hidden units per layer.

**Hyperparameter selection (grid sweeps).** We choose key hyperparameters by sweeping a small grid on the training indices in $\mathsf{Train}$ and selecting values that yield (i) high teacher specification satisfaction and (ii) stable Stage 2 template fitting. In particular, we sweep the candidate-pool sampling rates $(\alpha, \beta, \gamma)$ used to build $\mathcal{D}_i$, the confidence threshold $\tau_c$ used to filter candidates, and the confidence-loss weight $\lambda_{\mathrm{conf}}$ used in the joint objective $\mathcal{L}^{(i)}_{\mathrm{teacher-conf}} = \mathcal{L}^{(i)}_{\mathrm{teacher}} + \lambda_{\mathrm{conf}} \mathcal{L}^{(i)}_{\mathrm{conf}}$. Concretely, we sweep $\alpha \in \{0.6, 0.7, 0.8, 0.9, 1.0\}$, $\beta \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, and $\gamma \in \{0.6, 0.7, 0.8, 0.9, 1.0\}$, along with $\tau_c \in \{0.7, 0.8, 0.9\}$ and $\lambda_{\mathrm{conf}} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. We select the configuration that produces strong teachers and reliable filtering while retaining suffi-




(a) CAR2D (b) SIMPLEDRONE (c) DRONEATTITUDE

Figure 5: Agent models and their dynamics.

cient coverage for Stage 2. We similarly choose the cross-index regularization weight by sweeping $\lambda_{\text{x}} \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$ and selecting the largest value that reduces cross-index teacher drift without degrading teacher specification satisfaction.

**Dataset aggregation settings.** For each $i \in \mathsf{Train}$, we construct a preliminary candidate set $\mathcal{D}_i$ by sampling states from the converged teacher's on-policy rollouts, the teacher replay buffer, and environment resets using the sampling rates $(\alpha, \beta, \gamma)$. We then keep a candidate state $s \in \mathcal{D}_i$ only if $p_i^{\text{conf}}(s) \geq \tau_c$, query the teacher for an action label $a = \hat{\pi}_i(s)$, and add $(s, a)$ to the final dataset $\widetilde{\mathcal{D}}_i$ until reaching an aggregation budget of $M$ labeled state–action pairs per training index. We select $M$ via a sweep $M \in \{5 \times 10^2, 5 \times 10^3, 5 \times 10^4, 5 \times 10^5, 5 \times 10^6\}$.

**Stage 2 behavioral cloning settings.** We fit the template coefficients $\kappa$ using the squared-error BC loss over $\bigcup_{i \in \mathsf{Train}} \widetilde{\mathcal{D}}_i$ with a first-order optimizer. We use Adam with an annealed learning rate initialized at $\eta = 1 \times 10^{-1}$, batch size 512, and train until the BC loss stagnates. At each outer iteration, we unroll the current $\kappa$ from the base policy $\pi_0$ to obtain the template-generated policies $\{\pi_i\}_{i \in \mathsf{Train}}$ used to evaluate $\mathcal{L}_{\text{BC}}$.

**Training compute.** All experiments were run on a SLURM cluster with Intel Xeon Gold 6226 CPUs (2.7 GHz, 24 cores per node) and 192 GB RAM per node. The codebase is available at https://anonymous.4open.science/r/Imitation-Kappa-2094/.

# F Environment Description

Illustrations of the environments are given in Figure 5.

# G Specifications

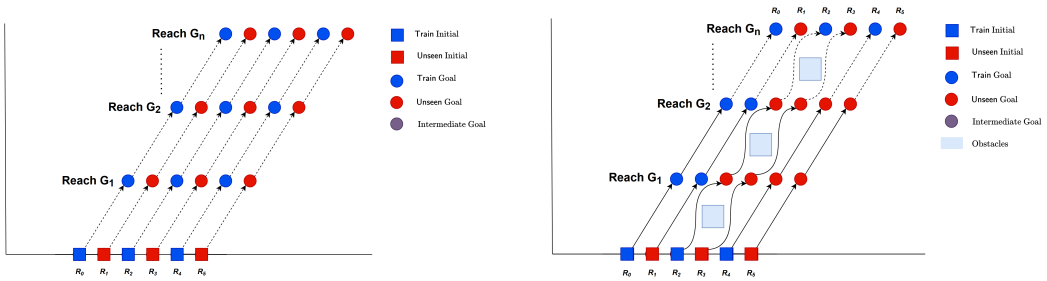
## G.1 $n$-REACHABILITY (Figure 6.a)

$$\texttt{achieve}\ \big(\texttt{reach}\ (g_1)\big)\ ;\ \texttt{achieve}\ \big(\texttt{reach}\ (g_2)\big)\ ;\ \cdots\ ;\ \texttt{achieve}\ \big(\texttt{reach}\ (g_n)\big)$$

This specification requires the agent to reach a sequence of $n$ target regions in a fixed order. The operator $\texttt{achieve}\ (\cdot)$ means the corresponding region must be reached at some point in the rollout, and the sequential composition ";" enforces ordering, i.e., the agent must reach $g_1$ first, then (after that) reach $g_2$, and so on, until it eventually reaches $g_n$.

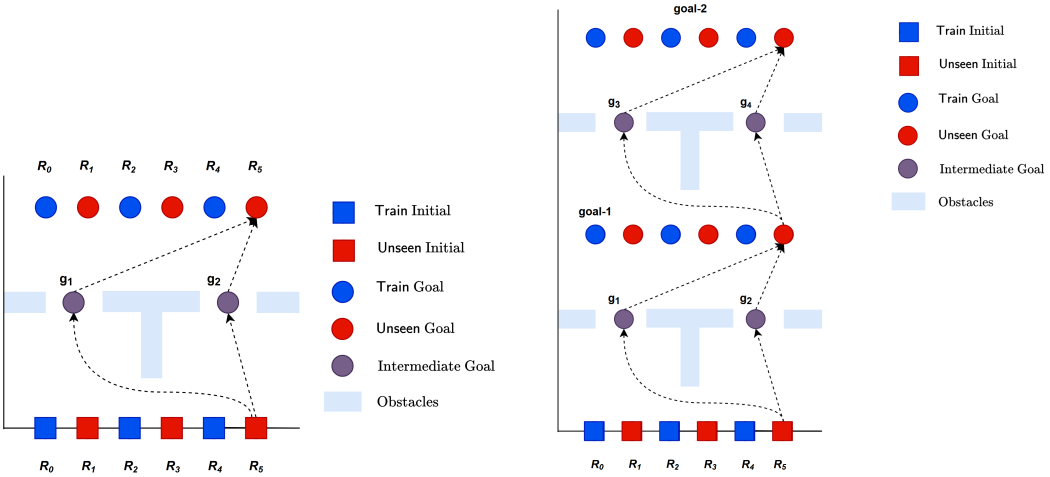
## G.2 $n$-REACHABILITY+OBS (Figure 6.b)

$$\texttt{achieve}\ \big(\texttt{reach}\ (g_1)\big)\ ;\ \texttt{achieve}\ \big(\texttt{reach}\ (g_2)\big)\ ;\ \cdots\ ;\ \texttt{achieve}\ \big(\texttt{reach}\ (g_n)\big)\ \texttt{ensuring}\ \big(\texttt{avoid}\ (obs)\big)$$




(a) $n$-REACHABILITY specification illustration. (b) $n$-REACHABILITY+OBS specification illustration.

Figure 6: Specification illustrations for $n$-REACHABILITY and $n$-REACHABILITY+OBS.




(a) CHOICE(L) specification illustration where $l = 1$ (b) CHOICE(L) specification illustration where $l = 2$

Figure 7: CHOICE(L) specification illustrations for two levels.

This specification adds a safety requirement to $n$-REACHABILITY. The first line is the same ordered sequence of reach goals. The second line, $\texttt{ensuring}\ (\texttt{avoid}\ (obs))$, requires that the agent avoid the obstacle region $obs$ at *all* timesteps during execution.

### G.3 CHOICE($l$) (Figure 7.b)

$$\left(\begin{array}{c}\texttt{achieve}\ \left(\texttt{reach}\ (g_{i1})\ \vee\ \texttt{reach}\ (g_{i2})\right)\ ; \\ \texttt{achieve}\ \left(\texttt{reach}\ (goal_i)\right) \\ \texttt{ensuring}\ \left(\texttt{avoid}\ (obs)\right)\end{array}\right)^{l}$$

This specification stacks $l$ levels of a branching subtask. At each level $i$, the agent must first reach *either* $g_{i1}$ or $g_{i2}$ (the disjunction $\vee$), and then reach the corresponding level goal $goal_i$. $(\cdot)^l$ indicates that this two-step pattern is repeated for $i = 1, \ldots, l$ in sequence. Finally, $\texttt{ensuring}\ (\texttt{avoid}\ (obs))$ enforces obstacle avoidance throughout the entire rollout.

### G.4 REACHER Specifications

Each task is a sequential reach specification $\texttt{achieve}\ (\texttt{reach}\ (\cdot)); \texttt{achieve}\ (\texttt{reach}\ (\cdot))$: in which the arm first reaches a designated "pick" region (typically the top block of the source tower), then reaches a designated "place" region (drop box or target tower).

| Baseline | Training tasks | Unseen tasks |
|---|---|---|
| ARS | – | – |
| BC | $3.94\times$ | – |
| MAML | $10.83\times$ | $14.04\times$ |
| VariBAD | $3.02\times$ | $3.04\times$ |
| GenRL | $3.82\times$ | $6.19\times$ |

Table 1: **Relative aggregate performance of DIBS.** Ratios compare the aggregate number of tasks across all benchmarks solved by DIBS against each baseline, using mean performance over **ten** random seeds. A dash indicates that the baseline solves zero tasks in the aggregate, so the ratio is undefined.

**Pick and drop (same side; Fig. 4.a).** Pick the top block from the source tower, then place it in the target drop box.

$$\texttt{achieve}\ \big(\texttt{reach}\ (\text{top block of source tower})\big)\ ;$$
$$\texttt{achieve}\ \big(\texttt{reach}\ (\text{target drop box})\big)$$

**Pick and drop (opposite side; Fig. 4.b).** Pick the top block from the source tower, then place it in the target drop box on the opposite side.

$$\texttt{achieve}\ \big(\texttt{reach}\ (\text{top block of source tower})\big)\ ;$$
$$\texttt{achieve}\ \big(\texttt{reach}\ (\text{target drop box})\big)$$

**Pick and vertical stack (same side; Fig. 4.c).** Pick the top block from the source tower, then stack it onto the top of the target tower.

$$\texttt{achieve}\ \big(\texttt{reach}\ (\text{top block of source tower})\big)\ ;$$
$$\texttt{achieve}\ \big(\texttt{reach}\ (\text{top block of target tower})\big)$$

**Pick and vertical stack (opposite side; Fig. 4.d).** Pick the top block from the source tower, then stack it onto the top of the target tower on the opposite side.

$$\texttt{achieve}\ \big(\texttt{reach}\ (\text{top block of source tower})\big)\ ;$$
$$\texttt{achieve}\ \big(\texttt{reach}\ (\text{top block of target tower})\big)$$

# H Additional Results

Table 1 summarizes the aggregate advantage of DIBS over each baseline across all benchmarks. For each baseline, we report the factor by which DIBS solves more tasks, separately for training tasks and unseen tasks, using mean performance over ten random seeds. Overall, DIBS achieves substantially higher training scalability than all baselines. On unseen tasks, DIBS also shows strong zero-shot generalization gains, solving $6.19\times$ as many tasks as GenRL, $3.04\times$ as many as VARIBAD, and $14.04\times$ as many as MAML. The dash entries indicate cases where the corresponding baseline solves zero tasks in the aggregate, so the ratio is undefined.

## H.1 Additional $k$-reachability results

We report the remaining $k$-reachability experiments for $k = 4$ down to $k = 1$, corresponding to decreasing horizon lengths, across CAR2D, SIMPLEDRONE, DRONEATTITUDE, and CAR2D with obstacles. These results are shown in Figures 9, 10, 11, and 12. We also include the CAR2D obstacle experiment for $k = 5$ in Figure 8.

Across these settings, the same overall trend holds: DIBS maintains higher training scalability and stronger zero-shot generalization as the task horizon varies, while the baselines degrade more rapidly as the number of training or unseen tasks increases. The exception is the $k = 1$ setting, where the task is sufficiently short-horizon and does not require long-horizon compositional reasoning. In this easier regime, VARIBAD also performs well and slightly outperforms DIBS.

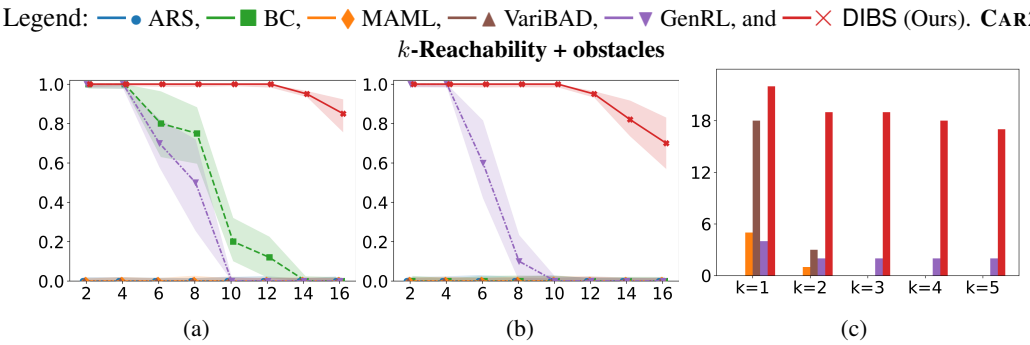



Figure 8: **Scalability plots for CAR2D $k$-Reachability with obstacles. Column (a)** shows the successful training ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 5$. **Column (b)** shows the zero-shot generalization ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 5$. **Column (c)** shows absolute zero-shot generalization ($y$-axis) for $k \in \{1, \ldots, 5\}$ ($x$-axis). We report mean ($\pm$ standard deviation) performance across **ten** random seeds.

### H.2 Additional Reacher results

Figure 14 reports the REACHER experiment with 9 blocks, complementing the 8- and 10-block settings shown in the main paper. The quantitative trends remain consistent: DIBS scales with the number of training indices and maintains strong zero-shot generalization to unseen configurations, while baselines either degrade in training scalability or fail to generalize as the environment difficulty increases.

### H.3 Choice benchmark results

We also include the CHOICE experiments for both one-level and two-level branching tasks in Figure 13. In these experiments, only GenRL and DIBS achieve non-trivial performance, since the other baselines do not have an explicit branching mechanism and therefore struggle to solve the structured choice problem. Across both branching levels, DIBS consistently outperforms GenRL, showing that the decoupled imitation-based template learning procedure is more effective for handling branching task structure and generalizing to unseen choice configurations.

## I Additional ablations

#### I.0.1 Teacher-driven subtask discovery enables branching around obstacles.

This ablation studies an obstacle setup in CAR2D that violates the single-template inductive bias of GenRL. Consider a family where the start region shifts monotonically along the $x$-axis while the goal lies above a central obstacle. For start indices on the left of the obstacle, the correct reach-avoid strategy routes around the *left* side; once the start crosses near the obstacle center, the correct strategy switches to routing around the *right* side. While the specification template is identical across indices (only the initial distribution is translated), the *policies* are not smoothly related across all indices, i.e., a contiguous set of indices must follow one side and the remaining indices must follow the other. GenRL, which fits a single template, regularizes toward one consistent mode and often collapses to a single route (e.g., always going left), failing to represent the required switch.

**Branched template fitting.** Our decoupled method enables a simple preprocessing step that leverages teacher behavior before template fitting. In this ablation, we first train near-optimal per-task teachers without cross-index regularization (i.e., we do not constrain teachers for different indices to have similar weights or shared evolution). Because each teacher is optimized only for its own task instance, the resulting rollouts naturally realize both routing modes (left-of-obstacle and right-of-obstacle) when those modes are required. We then use these rollouts as a preprocessing signal for *subtask discovery*. We cluster teacher trajectories (and thus their corresponding indices) into two groups, yielding a partition $\mathsf{Train} = \mathsf{Train}_{\text{left}} \cup \mathsf{Train}_{\text{right}}$.

Finally, we fit two separate templates, $\kappa^{\mathrm{left}}$ and $\kappa^{\mathrm{right}}$, each using teacher-labeled datasets restricted to its cluster. At inference time, we unroll the corresponding template in index space (as in Section D.1) to generate policies within that subfamily. Since each template only needs to model a single coherent routing mode, template fitting becomes easier and more accurate, and the resulting branched generator recovers the mode switch around the obstacle.

**Result.** Compared to fitting a single template over all indices, the branched variant avoids mode collapse and correctly switches routes around the obstacle as the start crosses the obstacle center, improving specification satisfaction in this setting.

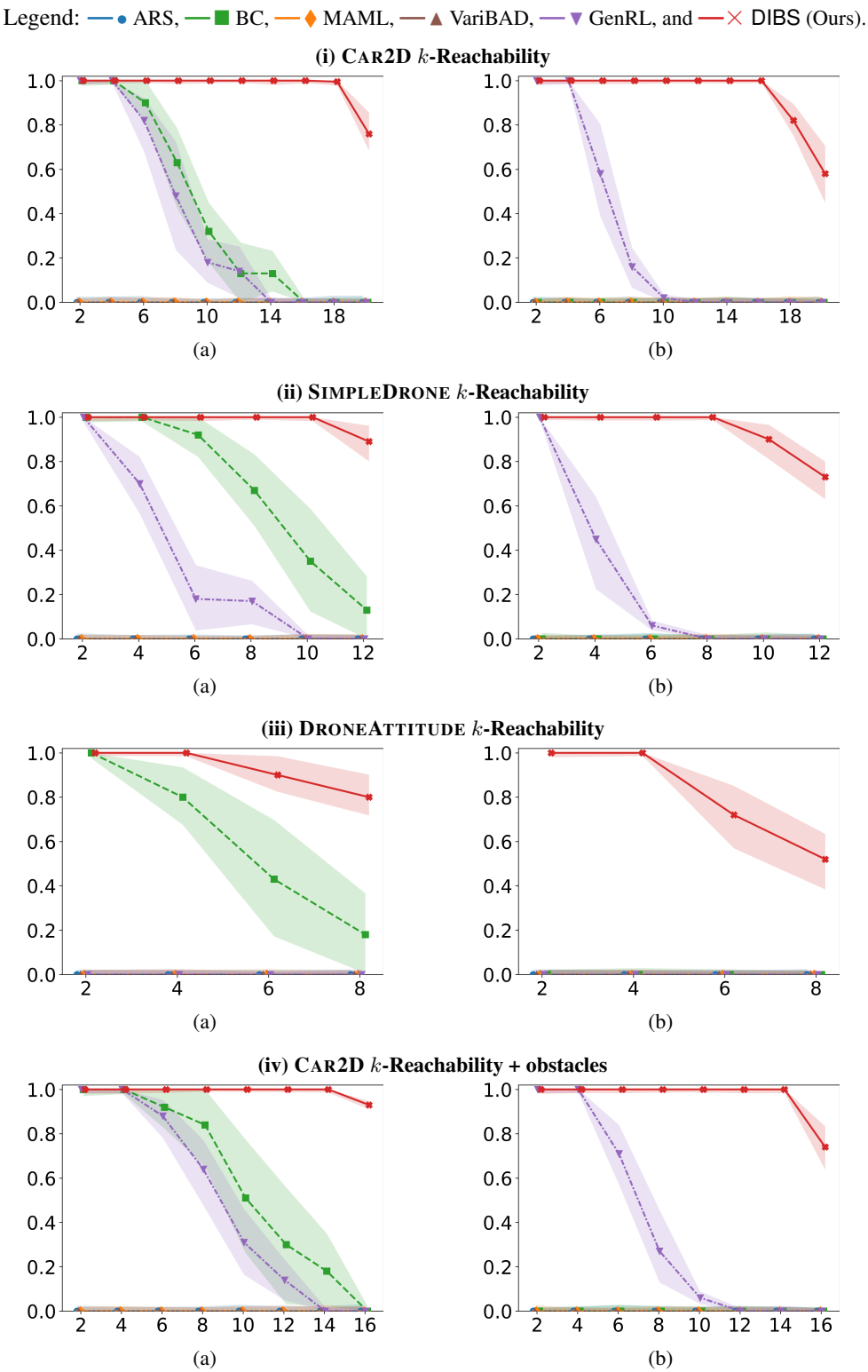



Figure 9: **Scalability plots for $k = 4$ reachability.** Each row corresponds to one environment on $k$-reachability tasks. **Column (a)** shows the successful training ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 4$. **Column (b)** shows the zero-shot generalization ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 4$. We report mean ($\pm$ standard deviation) performance across **ten** random seeds.

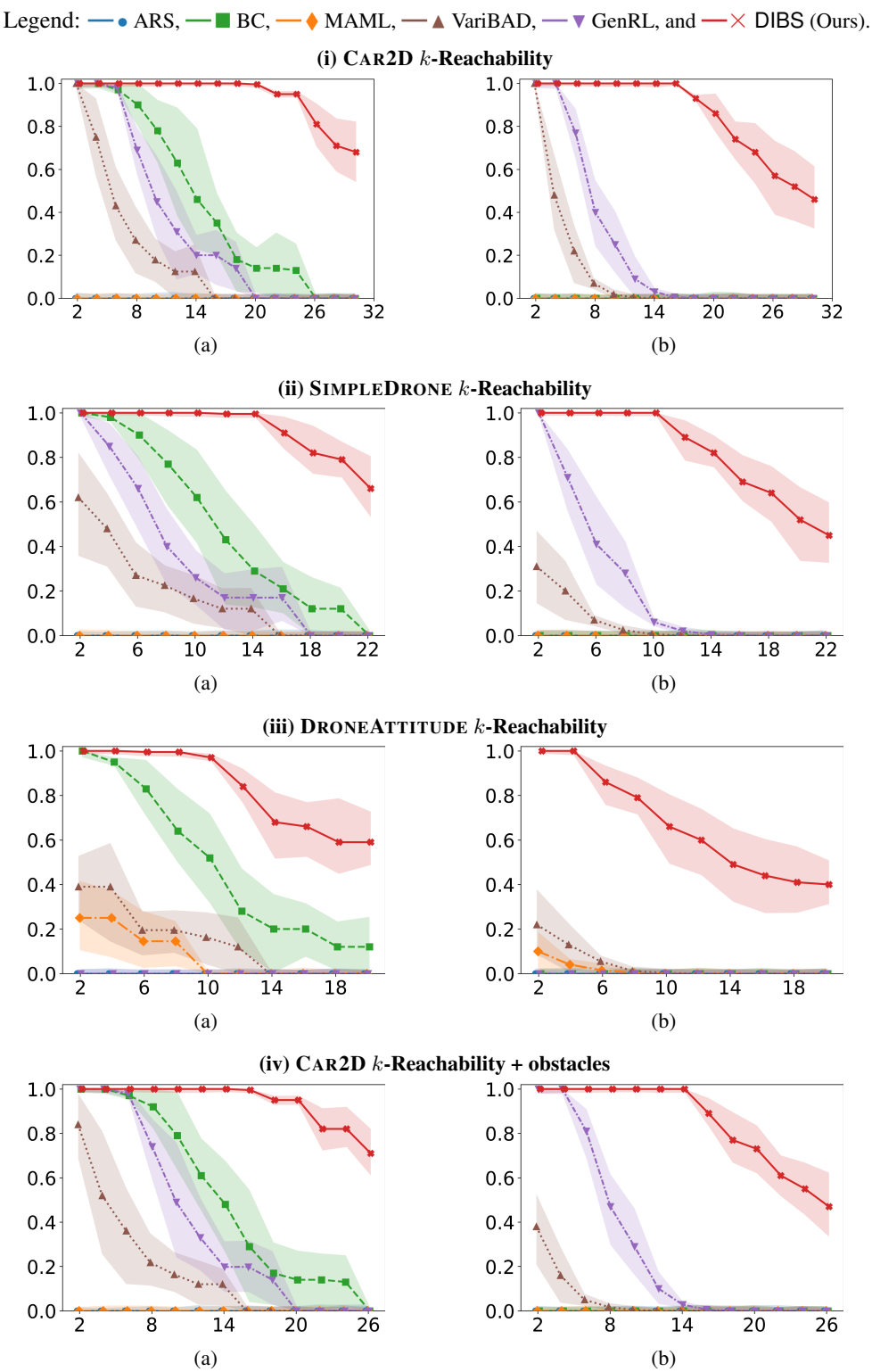



Figure 10: **Scalability plots for $k = 3$ reachability.** Each row corresponds to one environment on $k$-reachability tasks. **Column (a)** shows the successful training ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 3$. **Column (b)** shows the zero-shot generalization ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 3$. We report mean ($\pm$ standard deviation) performance across **ten** random seeds.

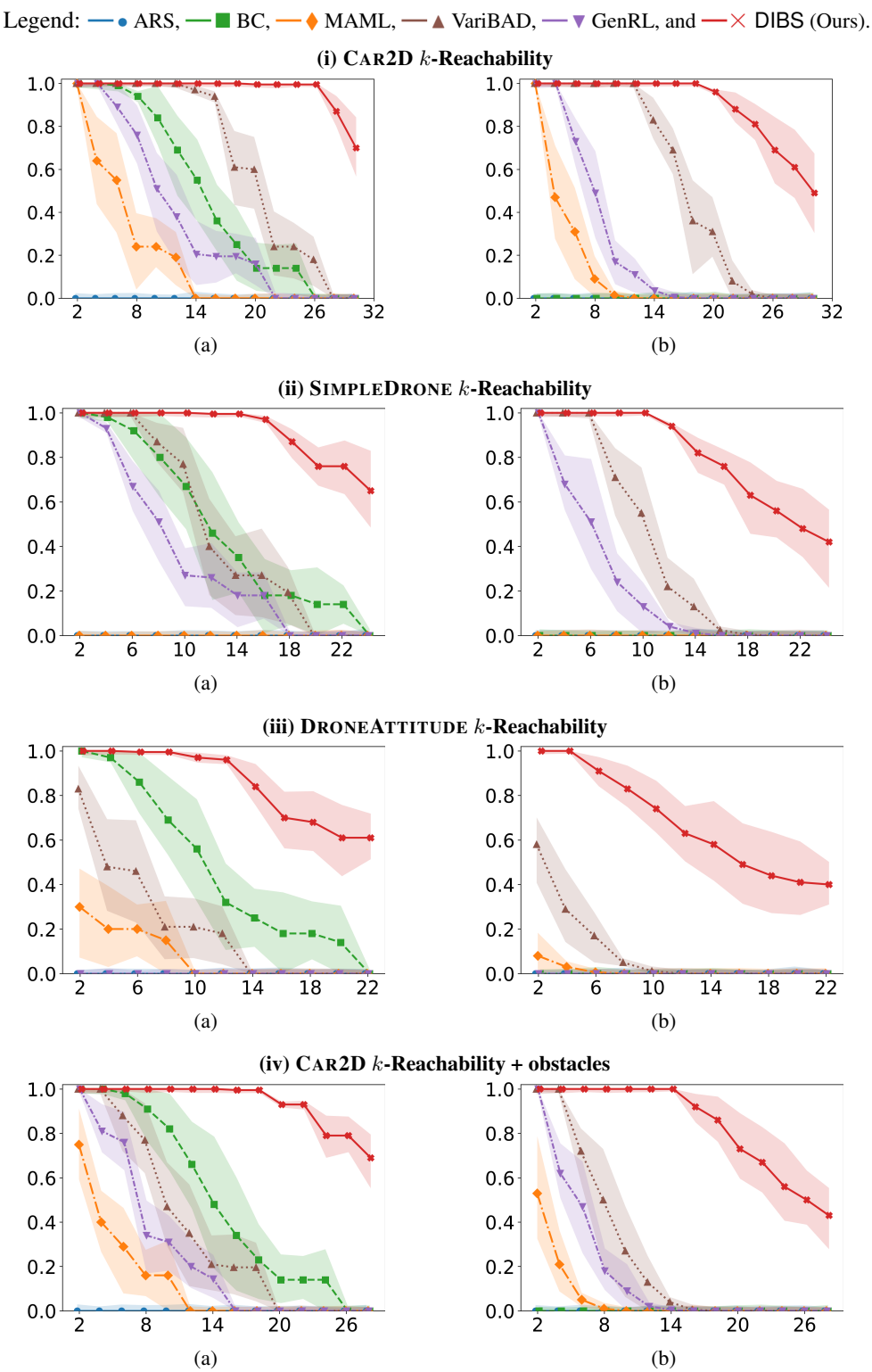



Figure 11: **Scalability plots for $k = 2$ reachability.** Each row corresponds to one environment on $k$-reachability tasks. **Column (a)** shows the successful training ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 2$. **Column (b)** shows the zero-shot generalization ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 2$. We report mean ($\pm$ standard deviation) performance across **ten** random seeds.

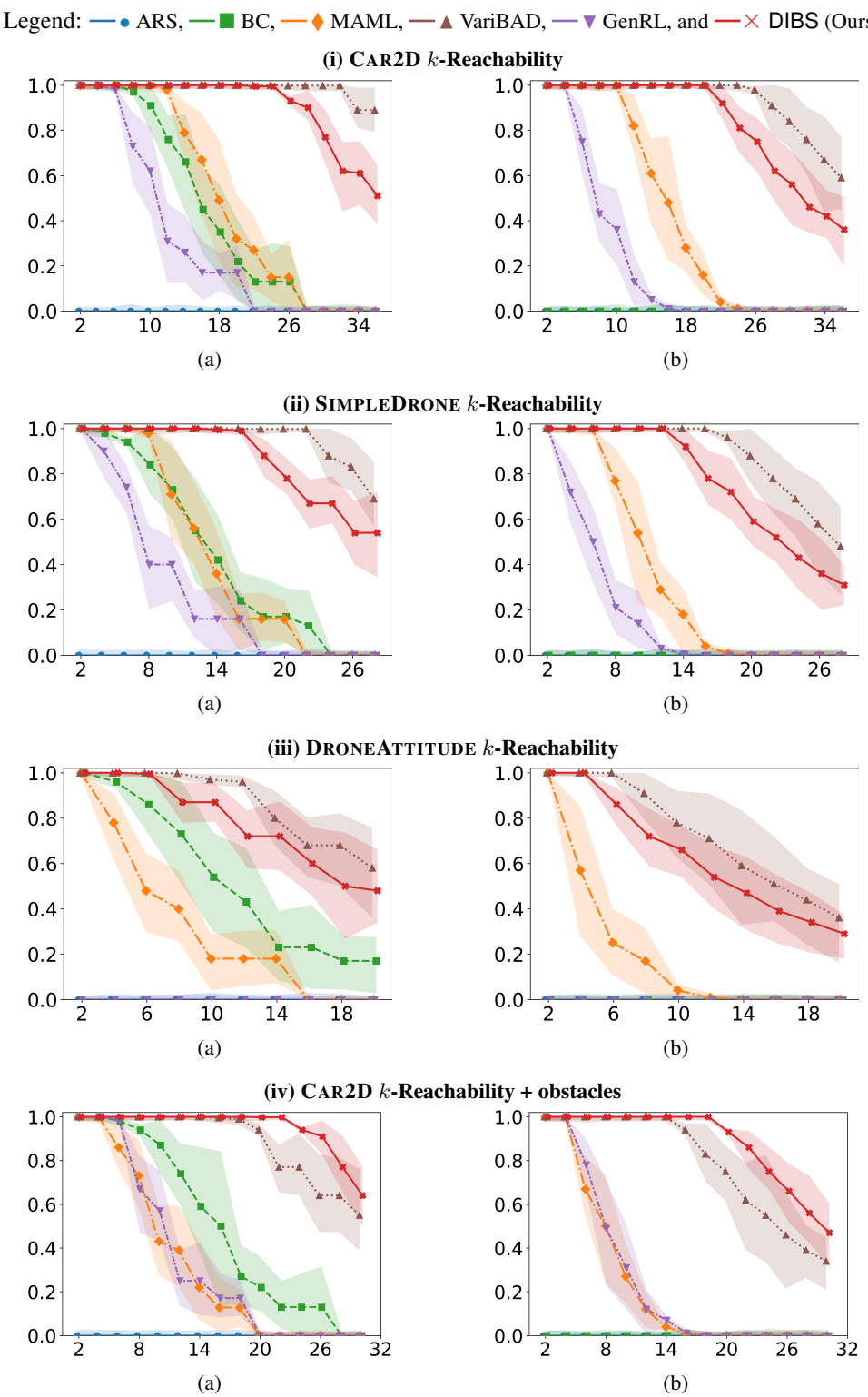



Figure 12: **Scalability plots for $k = 1$ reachability.** Each row corresponds to one environment on $k$-reachability tasks. **Column (a)** shows the successful training ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 1$. **Column (b)** shows the zero-shot generalization ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis) for $k = 1$. We report mean ($\pm$ standard deviation) performance across **ten** random seeds.

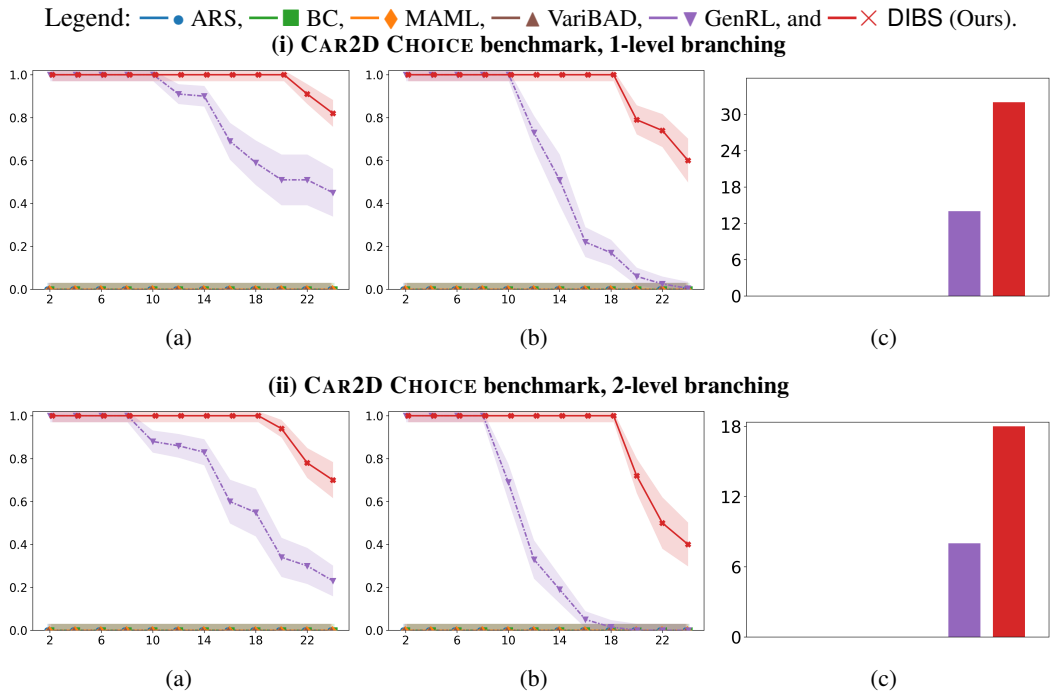



Figure 13: **Scalability plots for the CHOICE benchmark.** Each row corresponds to one CAR2D CHOICE benchmark variant. **Column (a)** shows the successful training ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis). **Column (b)** shows the zero-shot generalization ratio ($y$-axis) versus the number of training tasks $|\mathsf{Train}|$ ($x$-axis). **Column (c)** shows absolute zero-shot generalization ($y$-axis). We report mean ($\pm$ standard deviation) performance across **ten** random seeds.

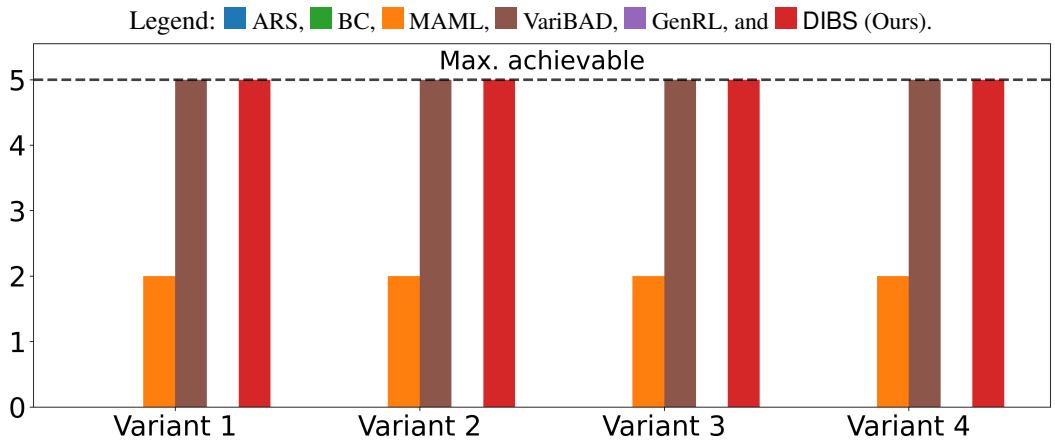



Figure 14: REACHER: 9 blocks