What to Represent, How to Act: Programmatic Feature Induction for Few-Shot Bayesian Imitation

Zahra Bashir[1,2], **Kelsey Allen**[2,†], **Tom Silver**[1,†]
zb2882@princeton.edu, krallen@cs.ubc.ca, tsilver@princeton.edu

[1]**Princeton University**
[2]**University of British Columbia**
[†]Equal co-advisors.

Abstract

A natural way to teach an agent a new task is to give it a few expert demonstrations. But few-shot imitation requires more than reproducing observed state–action pairs: the agent must infer an underlying policy that generalizes across task instances. We study a controlled setting where the relevant abstractions are not known in advance, as a step toward big-world settings where the same problem is much harder due to partial observability, continuous control, and compounding error. We propose TwoLIP: Two-level LLM-Guided Induction of Programmatic Policies. At the first level, TwoLIPuses a large language model (LLM) to build a library of programmatic state–action features. At the second level, it performs Bayesian implicit behavioral cloning (BC) to learn a programmatic policy that combines the features. This two-level pipeline separates the problem of what to represent from how to act, combining a semantic prior over candidate features with a structural prior over policy composition. We evaluate TwoLIPon Generalization Grid Games, a suite of strategy tasks where policies must generalize to unseen configurations that differ substantially from the demonstrations. TwoLIPconsistently outperforms BC baselines, including deep neural models and end-to-end LLM-based policy-generation approaches. We also find preliminary evidence that induced feature libraries transfer across related tasks. Together, these results point toward a path for few-shot imitation in big worlds, where the right abstractions cannot be prespecified.

1 Introduction

In large, open-ended worlds, agents will encounter tasks that were not anticipated when their policy representations were designed. Consider a household robot learning from a few demonstrations to organize an unfamiliar kitchen counter: the user may move a mug beside the coffee machine or place a spoon in a drawer. What transfers is not a trajectory or visual template, but semantic distinctions such as which object is out of place and where it belongs. This example reflects a central challenge in modern imitation learning (IL): demonstrations provide a natural interface for teaching agents, but a few demonstrations rarely specify the policy uniquely. In big worlds, this ambiguity is amplified, because agents must generalize across changing objects, layouts, and task instances where the relevant abstractions are not known in advance.

Standard BC treats imitation as supervised action prediction: given an observation, predict the demonstrated action (Pomerleau, 1988). Recent advances in robot learning have scaled this paradigm to increasingly complex behaviors from demonstration data (Zhao et al., 2023; Chi et al., 2025; Khazatsky et al., 2024). A related line of work on implicit BC instead scores candidate state–action pairs and has shown strong results in robot policy learning (Florence et al., 2022). However, both explicit and implicit BC remain representation-dependent: with very limited data, they can

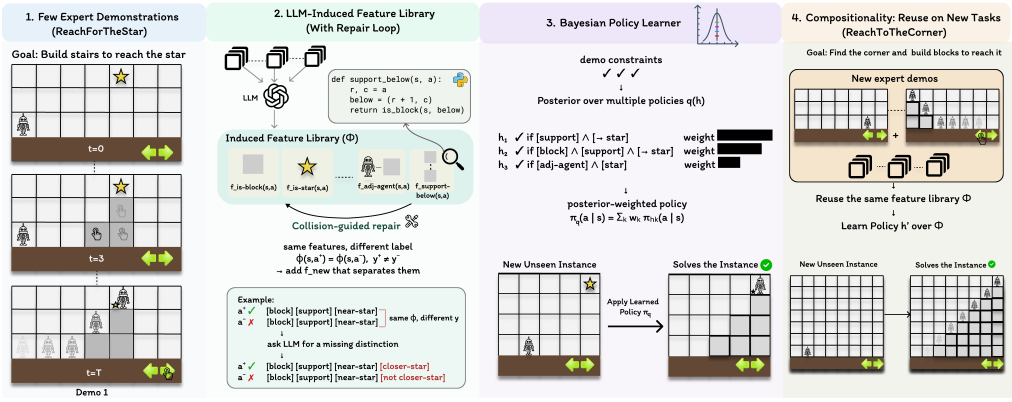


Figure 1: Overview of TwoLIP. From a few expert demonstrations, an LLM induces a reusable feature library, a Bayesian learner composes these features into a logical policy, and the same library can be reused for related tasks.

rely on brittle patterns unless the representation captures the semantic distinctions needed to choose among actions (Pari et al., 2022).

Structured symbolic and programmatic approaches address this representation problem through a strong inductive bias, expressing policies in terms of rules or programs (Verma et al., 2018; Silver et al., 2020; Xin et al., 2023; Patton et al., 2023; Cui et al., 2025), but typically assume that the relevant predicates or domain-specific languages (DSLs) are provided in advance. LLMs offer a promising way to relax this assumption: Code as Policies (CAP) and similar lines of work show that LLMs can generate executable policy code from language and examples (Liang et al., 2023). However, as we show in our experiments, directly asking an LLM to infer a complete policy from a few demonstrations can be brittle in complex environments.

To address these challenges, we propose TwoLIP—Two-level LLM-Guided Induction of Programmatic Policies—which separates the problem of *what to represent* from *how to act*. Rather than asking an LLM to generate a complete policy, TwoLIPuses the LLM to induce and refine programmatic state–action features (*what to represent*). TwoLIPthen performs implicit BC over this induced feature space (*how to act*). A further benefit of this two-level pipeline is that it admits a natural feedback loop between feature generation and classification. We begin with an initial feature set and identify positive and negative state–action pairs that the current features cannot separate. These pairs are fed back to the LLM to guide targeted feature generation, and the process repeats until all pairs are separable (or a maximum number of rounds is reached). To maintain interpretability and quantify uncertainty, we use an implicit BC technique that is programmatic and Bayesian (Silver et al., 2020). See Figure 1 for an overview of the full method.

We evaluate TwoLIPon Generalization Grid Games (Silver et al., 2020), a suite of strategy tasks that test generalization from a few demonstrations to substantially different held-out game instances. TwoLIPachieves higher task-completion success than supervised BC and end-to-end LLM policy-generation baselines in the low-data regime, while approaching the performance of an oracle that uses a hand-designed feature grammar. We also find preliminary evidence that induced feature libraries can be reused across related tasks without retraining, suggesting that the learned representations capture reusable task structure rather than demonstration-specific correlations. Together, these results suggest that decoupling representation discovery from policy learning is a promising path toward few-shot imitation in big worlds, where the right abstractions cannot be specified in advance.

2 Problem Setting

We consider IL in a finite-horizon Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \rho_0, \mathcal{G}, H)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s' \mid s, a)$ is the transition kernel, ρ_0 is a distribution over

initial states, $\mathcal{G} \subseteq \mathcal{S}$ is the set of goal states, and H is the horizon. The learner is given a small set of expert demonstrations $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$, where $\tau_i = (s_0^i, a_0^i, s_1^i, a_1^i, \ldots, s_{T_i}^i)$, $s_t^i \in \mathcal{S}$, $a_t^i \in \mathcal{A}$, and N is small (e.g., $N \leq 10$ in our experiments). The demonstrations are generated by an expert policy

$$\pi^* : \mathcal{S} \to \mathcal{A}, \qquad a_t^i = \pi^*(s_t^i),$$

which induces successful behavior in $\mathcal{M}$. We assume demonstrations come from a consistent expert with a canonical action for each demonstrated state, though not necessarily a uniquely optimal one: if multiple actions are equally optimal, each is valid. The goal is to learn a policy $\pi \in \Pi$, with $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, from $\mathcal{D}$ that imitates the expert and generalizes beyond the observed trajectories. We evaluate a learned policy by rolling it out from held-out initial states $s_0 \sim \rho_{\text{test}}$ and measuring task-completion success:

$$J(\pi) = \Pr_{\substack{s_0 \sim \rho_{\text{test}}, \\ a_t \sim \pi(\cdot|s_t), \\ s_{t+1} \sim P(\cdot|s_t, a_t)}} \left[\exists t \leq H \text{ such that } s_t \in \mathcal{G}\right].$$

3 Method

Overview. TwoLIPseparates the few-shot imitation problem into two levels. First, it constructs a library of semantic state–action features: reusable Boolean functions of a candidate action in the current state. Second, it performs Bayesian inference over structured policies that combine these features to decide which actions are admissible. The remainder of this section describes the two levels of TwoLIP. We first define the behavior-learning problem by converting demonstrations into a contrastive state–action dataset, which provides positive and negative examples for the learning task (Section 3.1). We then introduce the induced policy class and Bayesian inference procedure (Sections 3.2 and 3.3). Finally, we describe how TwoLIPconstructs the feature library from demonstrations and uses feature collisions to diagnose and repair missing abstractions (Sections 3.4, 3.5, and 3.6). Algorithm 1 summarizes the pipeline, and Figure A.1 visualizes a learned policy.

3.1 Contrastive State–Action Imitation Data

We convert demonstrations into a labeled contrastive dataset over state–action pairs. Each demonstrated action a_t^i at state s_t^i gives a positive example $(s_t^i, a_t^i, 1)$. To construct negatives, we select alternative actions from a candidate set $\mathcal{C}^-(s_t^i, a_t^i) \subseteq \mathcal{A} \setminus \{a_t^i\}$. This set may contain all non-expert actions or only a sampled subset. The resulting dataset is $\mathcal{D}_{\text{sa}} = \{(s, a, y)\}$, with $y = 1$ for expert actions and $y = 0$ for alternatives. This turns imitation into a state–action discrimination problem: the learner compares the expert action against other actions available in the same state, assigning high admissibility, or low energy, to expert actions and lower admissibility to alternatives (Florence et al., 2022; LeCun et al., 2006).

3.2 Feature-Induced Implicit Policies

A feature is a Boolean function over state–action pairs. Given a feature library $\Phi = \{\phi_1, \ldots, \phi_m\}$, each state–action pair is mapped to a Boolean vector $x_\Phi(s, a) = (\phi_1(s, a), \ldots, \phi_m(s, a)) \in \{0, 1\}^m$. TwoLIPlearns a structured classifier $h : \{0, 1\}^m \to \{0, 1\}$, where $h(x_\Phi(s, a)) = 1$ indicates that a is consistent with the expert at state s and $h(x_\Phi(s, a)) = 0$ indicates that it is not. This defines an implicit policy over candidate actions: rather than predicting an action directly from the state, h evaluates each candidate state–action pair through the induced feature representation. The test-time decision rule, which uses the posterior over classifiers, is given in Section 3.3. To keep the learned policy human-interpretable, we restrict h to logical rules over the induced features. Concretely, policies are represented as disjunctions of conjunctions of feature literals, $h(x_\Phi(s, a)) = \bigvee_r \bigwedge_{j \in I_r} \ell_{rj}(s, a)$, where $I_r \subseteq \{1, \ldots, m\}$ and $\ell_{rj}(s, a) \in \{\phi_j(s, a), \neg\phi_j(s, a)\}$. Each conjunction gives one sufficient condition for an action to be admissible, while the disjunction combines multiple such reasons. In practice, we instantiate this policy class using decision trees over

Boolean features (Breiman et al., 1984): each path to a positive leaf corresponds to one conjunction, and the set of positive leaves defines the disjunction.

3.3 Bayesian Inference over Structured Policies

The policy-composition layer builds on the Bayesian logical policy-learning formulation of Logical Program Policies (LPP) (Silver et al., 2020). This Bayesian view is useful in our few-demonstration setting because the data often underdetermines the policy: many classifiers may explain the observed state–action labels, and committing to a single one can overfit to accidental patterns in the demonstrations. We therefore maintain an approximate posterior over structured policies. Let $\mathcal{H}_\Phi$ denote the hypothesis class of structured classifiers over the induced feature representation x_Φ. Each candidate policy $h \in \mathcal{H}_\Phi$ receives a posterior score that combines its agreement with the contrastive dataset $\mathcal{D}_{\text{sa}}$ and a structural prior favoring simpler policies. In particular, policies that mark expert actions as admissible and alternatives as inadmissible receive higher data likelihood, while more complex policies receive lower prior probability.

Exact inference over $\mathcal{H}_\Phi$ is intractable, so we approximate the posterior with a weighted collection of policy particles $q(h) = \sum_{k=1}^{K} w_k \delta_{h_k}(h)$, where each h_k is a structured classifier over the induced feature space and w_k is proportional to its posterior score. At test time, the posterior mixture assigns each candidate action a soft admissibility score $S_q(s, a) = \mathbb{E}_{h\sim q}[h(x_\Phi(s, a))] = \sum_{k=1}^{K} w_k h_k(x_\Phi(s, a))$. The policy then selects an action with maximum posterior expected admissibility, $\pi_q(s) \in \arg\max_{a\in\mathcal{A}} S_q(s, a)$. This posterior mixture acts as a soft energy-like state–action scoring model and is especially useful in the few-demonstration setting, where several distinct policies may be consistent with the observed behavior. Additional details are in Appendix A.

3.4 Demo-Guided State–Action Feature Induction

The policy learner above assumes a feature library Φ, but in real-world problems the relevant abstractions are often not known in advance. We therefore do not assume access to an oracle set of human-designed features, DSL primitives, or grammar rules tailored to the task. Instead, TwoLIP-constructs this library automatically from demonstrations using an LLM. The prompt template used for feature induction is provided in Appendix D.1.

Rather than asking the LLM to synthesize the full policy, we ask it to generate feature-template generators: small Python functions that define families of state–action features. For example, in a grid task, a template may check whether the candidate action selects an object of type c, or whether it moves toward an object with property p. These templates are expanded into candidate Boolean features. Before adding candidates to the library, we evaluate them on $\mathcal{D}_{\text{sa}}$ and apply a *feature-filtering* phase. This removes features that fail to execute, violate the required type signature, are constant on the dataset, or duplicate the behavior of an existing feature on the observed examples. The surviving features form the initial library $\Phi^{(0)} = \{\phi_1, \ldots, \phi_m\}$, where the superscript 0 denotes the library before any repair rounds.

3.5 Detecting Feature Insufficiency with Collisions

The induced feature library determines what distinctions the policy learner can express. To determine whether the current feature set can separate expert actions from alternatives, we introduce the concept of **feature collisions**. A collision occurs when two labeled examples have the same feature representation but different labels: $x_\Phi(s, a) = x_\Phi(s', a') \quad \text{and} \quad y \neq y'$.

When this happens, under the consistency assumption from Section 2, no deterministic classifier whose prediction depends only on x_Φ can correctly classify both examples. Thus, mixed-label collisions identify an irreducible source of empirical error for any policy expressed using the current feature library. To measure this effect, we group examples in $\mathcal{D}_{\text{sa}}$ into *feature buckets*, where each bucket contains all examples with the same feature vector $x_\Phi(s, a)$. A bucket is mixed-label if it

contains both expert actions and alternatives. We quantify collision severity by counting either the number of mixed-label buckets or the number of positive–negative pairs within them.

This diagnostic is one of the key distinctions of our framework. Because feature induction and policy learning are separated, the feature library is an explicit intermediate object that can be inspected, filtered, and repaired before learning the final policy.

3.6 Collision-Guided Feature Repair

TwoLIPuses feature collisions as targeted feedback for improving the library. At repair round r, after constructing the current feature set $\Phi^{(r)}$, we identify mixed-label buckets in $\mathcal{D}_{\text{sa}}$ and sample representative[1] positive–negative collision pairs from them. These pairs are given to the LLM as counterexamples showing where the current abstraction fails, and the LLM is prompted to propose additional predicates that distinguish the expert action from the alternative. Appendix D.2 provides the full prompt and implementation details.

The newly proposed features are instantiated and optionally filtered. We accept them only if they reduce the collision-induced empirical-error bound; otherwise, we discard them and keep the library unchanged. Accepted features are added as $\Phi^{(r+1)} = \Phi^{(r)} \cup \Delta\Phi^{(r)}$. We then recompute collisions and repeat until the repair budget is exhausted or no collisions remain. The final library is passed to the policy learner. The next proposition formalizes the effect of an accepted repair step.

Proposition 1 (Collision-added features strictly reduce irreducible empirical error)**.** *Let Φ be a feature library and let Ψ be a set of additional features added by an accepted repair step. Define*

$$R^*(\Phi) = \min_{h:\{0,1\}^{|\Phi|}\to\{0,1\}} \frac{1}{|\mathcal{D}_{\text{sa}}|} \sum_{(s,a,y)\in\mathcal{D}_{\text{sa}}} \mathbf{1}\{h(x_\Phi(s,a)) \neq y\},$$

the minimum empirical classification error achievable by any deterministic classifier whose prediction depends only on $x_\Phi(s,a)$. Then

$$R^*(\Phi \cup \Psi) < R^*(\Phi).$$

A proof is given in Appendix B.

Collision-freeness should be interpreted carefully. If no collisions remain, then $\mathcal{D}_{\text{sa}}$ is separable at the representation level: there exists some deterministic classifier over the observed feature vectors that can fit the labels. This does not guarantee generalization, nor does it guarantee that a restricted or regularized policy learner will choose a perfect classifier. However, the presence of collisions is a concrete certificate that the current feature library is insufficient for fitting the labeled examples. More discussion about the expressiveness vs generalizability trade-off is written in Appendix F.

4 Experiments and Results

We evaluate TwoLIPalong three main axes: (i) generalization to unseen task instances, (ii) sample efficiency with respect to the number of demonstrations, and (iii) composability of the learned feature library across tasks.

4.1 Experimental Setup

Environments and Tasks We evaluate TwoLIPon Generalization Grid Games (Silver et al., 2020), a suite of strategy tasks with a shared grid-based state and action space. Each state is a variable-sized $h \times w$ grid with cell values from a finite set V, and each action selects a grid cell, $a \in \{1,\ldots,h\} \times \{1,\ldots,w\}$. We consider five tasks: *TwoPileNim*, *CheckmateTactic*, *StopTheFall*, *ReachForTheStar*,

[1] We rank mixed-label buckets by positive–negative pair count and sample from the top-ranked buckets.

Algorithm 1: TwoLIP

Require: demos $\mathcal{D}$, negative sampler C^-, query budget Q, rounds R, particles K
Ensure: feature library Φ, posterior q

$\mathcal{D}_{\text{sa}} \leftarrow \textsc{BuildData}(\mathcal{D}, \mathsf{C}^-)$
$\Phi \leftarrow \textsc{InduceFeatures}(\mathcal{D}_{\text{sa}}; Q)$
for $r = 1, \ldots, R$ **do**
  $(X, y) \leftarrow \textsc{Featurize}(\mathcal{D}_{\text{sa}}, \Phi)$
  $\mathcal{B} \leftarrow \textsc{FindCollisions}(X, y)$
  if $\mathcal{B} = \emptyset$ **then break**
  end if
  $\Phi \leftarrow \Phi \cup \textsc{RepairFeatures}(\mathcal{B})$
end for
$(X, y) \leftarrow \textsc{Featurize}(\mathcal{D}_{\text{sa}}, \Phi)$
$q \leftarrow \textsc{InferPosterior}(X, y, \Phi; K)$
return Φ, q

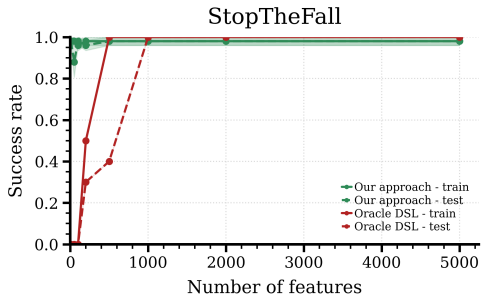


Figure 2: Task-completion success on STF as a function of feature-library size.

and *Chase*[2], each with different objectives and strategic requirements. For each task, instances vary in grid size, layout, and object configurations, requiring generalization beyond the training demonstrations. We use 10 training instances and 10 held-out test instances per task, with a 200-step episode horizon. Our primary metric is test success rate: the fraction of held-out instances solved within this horizon. Unless otherwise stated, all methods that rely on LLM calls use GPT-4.1 (OpenAI, 2025) with temperature 0.0 and top-p 1.0.

4.2 Baselines

We compare TwoLIPagainst baselines for IL and policy representation.

Fully Convolutional Network (FCN). We train an FCN (Long et al., 2015) using the same state and action representations as TwoLIP: eight 3×3 convolutional layers with stride 1, padding 1, and ReLU nonlinearities, exploiting local spatial structure through parameter sharing.

CAP + Demos. We evaluate an end-to-end LLM baseline inspired by Code as Policies (Liang et al., 2023), prompting the model to synthesize a Python policy from demonstrations. Unlike TwoLIP, it infers task structure directly from demonstrations without an explicit feature representation (Austin et al., 2021; Chen et al., 2021).

VLM-Imitation. We prompt a vision-language model with demonstration image sequences to predict actions for new visual states, testing whether multimodal models can perform IL directly without structured features.

Oracle DSL Baseline. We include a structured oracle baseline based on the LPP setup (Silver et al., 2020). This baseline uses a hand-designed DSL tailored to the tasks, so the policy learner already has access to task-relevant symbolic features. TwoLIPdoes not assume accessing this DSL; it has to induce its feature library from demonstrations. We use this comparison to see how close learned features can get to a structured policy learner that is given the right abstractions in advance.

4.3 Generalization Performance

We first evaluate whether TwoLIPgeneralizes from few demonstrations. Figure 3 reports test success versus the number of training demonstrations on 10 held-out task instances, excluded from both feature generation and policy learning. Shaded regions show standard error over 10 random seeds; the training curves are in Appendix C.

Across tasks, TwoLIPsubstantially outperforms the end-to-end baselines and is often competitive with the oracle DSL baseline. In some tasks, it matches or exceeds the oracle despite not using

[2]Short forms: nim, ct, stf, rfts, and ec, respectively.

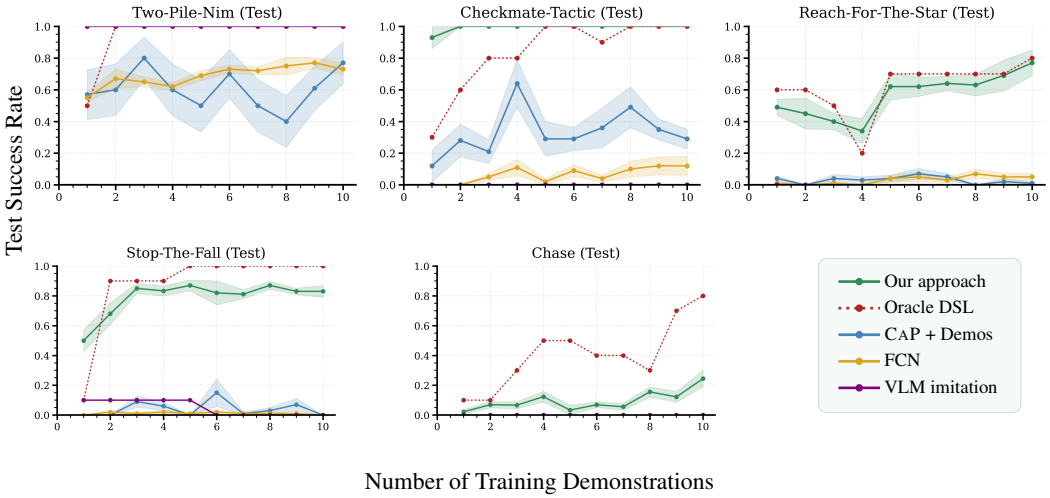
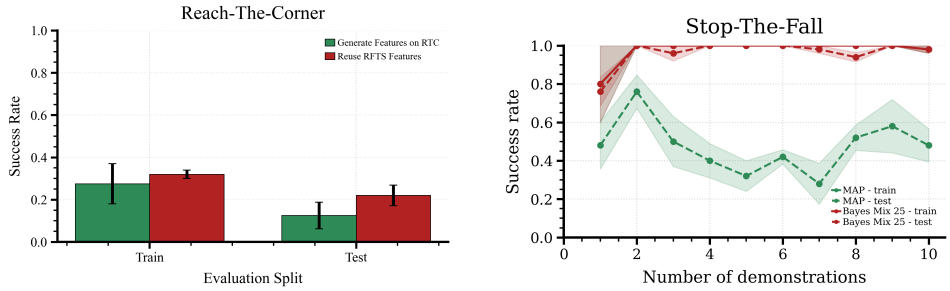


Figure 3: Performance on held-out test task instances as a function of the number of training demonstrations. Each subplot reports mean task-completion success over 10 random seeds.



(a) Transferability on RTC with 10 demos. (b) Performance of MAP vs. mixture policy.

Figure 4: Feature-transfer and posterior-mixture ablation results.

the hand-designed DSL. The FCN baseline improves slightly with more demonstrations but remains well below TwoLIP, suggesting that direct supervised action prediction needs more data to learn the relevant spatial patterns. The LLM-based baselines also perform poorly on most tasks, likely because direct policy synthesis requires inferring task logic and reliable spatial reasoning from text prompts. The main exception is *TwoPileNim*, whose state structure is simpler than the grid-navigation tasks. In one harder domain, TwoLIPremains below the oracle, likely due to the task difficulty and the expert strategy, but still outperforms baselines that achieve near-zero success.

Overall, these results show that separating feature induction from policy learning lets TwoLIPrecover task structure strong enough to match, and sometimes exceed, a policy learner given a hand-designed DSL, while substantially outperforming end-to-end imitation baselines.

4.4 Representation Efficiency

We next compare how much representation size is needed to reach strong performance. The oracle DSL baseline enumerates features from a fixed hand-designed DSL, independent of the demonstrations. In contrast, TwoLIPinduces features conditioned on the demonstrated behavior, so the feature library can focus on distinctions that are useful for the current task. Figure 2 shows that TwoLIPreaches performance comparable to the oracle DSL baseline while using far fewer features, often by one to two orders of magnitude depending on the environment. This compactness advantage holds for both training and held-out test instances, suggesting that TwoLIPinduces a compact, task-relevant feature library rather than relying on a large hand-enumerated representation.

4.5 Composability of Learned Features

We next test whether induced feature libraries transfer across related tasks. As a preliminary experiment, we introduce Reach-The-Corner (RTC), a new task in the same environment as RFTS; details are in Appendix G. We compare an RTC-specific library to reusing the fixed RFTS library. Figure 4a shows that the reused library performs close to the RTC-specific one on both train and held-out test instances, suggesting that induced features capture reusable environment structure rather than only task-specific correlations. Here we transfer only the initial library without repair; future work will study when transferred libraries need repair.

4.6 Ablation Studies

We ablate two components contributing to TwoLIP's performance. First, we compare the full posterior ensemble to the MAP policy. The mixture improves robustness, especially with few demonstrations, by averaging over multiple plausible policies; Figure 4b shows this result. Second, we study collision repair: increasing the repair budget improves train/test accuracy while reducing positive–negative collisions, indicating that the initial feature library misses task-relevant distinctions. Performance becomes near-perfect before all collisions are removed, suggesting that repair mainly resolves the ambiguities most relevant to the downstream learner. Figure A.3 reports this ablation.

5 Related Work

IL for robot policies. Recent robot learning increasingly uses large-scale IL to train policies directly from demonstrations. Diffusion policies model multimodal action distributions for visuomotor manipulation (Chi et al., 2025), while generalist robot policies and vision-language-action (VLA) models leverage broad robot datasets and language/vision pretraining to generalize across tasks and embodiments (Open X-Embodiment Collaboration et al., 2024; Octo Model Team et al., 2024; Kim et al., 2024; Black et al., 2025). These works show the benefits of scale. In contrast, TwoLIPinduces compact state–action features and learns a structured policy over them from few demonstrations.

Data-efficient IL. Prior work on data-efficient IL includes apprenticeship and inverse reinforcement learning, which infer rewards or costs from demonstrations before solving for a policy (Abbeel and Ng, 2004), and meta-imitation, which learns across task distributions to adapt from one or a few demonstrations at test time (Duan et al., 2017; Finn et al., 2017a;b). While effective, these methods typically rely on many related tasks and demonstrations during meta-training. We instead study few-shot target-task imitation, where abstractions must be inferred directly from demonstrations.

Structured and programmatic policy learning. Another way to improve generalization is to add symbolic, relational, or programmatic structure to the policy class. Relational RL and IL use logical representations to generalize across objects and states (Džeroski et al., 2001; Natarajan et al., 2011), while programmatic policy methods search over interpretable programs (Verma et al., 2018). LPP shows that Bayesian inference over symbolic policy structure can support few-shot generalization when a task-relevant DSL is available (Silver et al., 2020). These approaches show the value of structure, but usually assume that the relevant DSL or features are provided by a human designer. TwoLIPkeeps the structured policy bias, but induces the feature library from demonstrations.

Language models and executable policy structure. Recent work uses language models to generate executable code for embodied control, most notably CAP (Liang et al., 2023). These methods produce structured control logic but typically generate the policy directly. In contrast, we use the LLM to propose candidate state–action features, which are executed, filtered, repaired, and passed to a separate policy learner. This separation lets us test and improve the representation before committing to a policy. [3]

[3] Discussion and future work are in Appendix E.

References

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, page 1. ACM, 2004. doi: 10.1145/1015330.1015430.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 17–40, 2025.

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10–11):1480–1501, 2025. doi: 10.1177/02783649241273668.

Guofeng Cui, Yuning Wang, Wensen Mao, Yuanlin Duan, and He Zhu. Abstraction refinement-guided program synthesis for robot learning from demonstrations. *Proceedings of the ACM on Programming Languages*, 9(OOPSLA2):555–583, 2025. doi: 10.1145/3763070.

Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1–2):7–52, 2001. doi: 10.1023/A:1007694015589.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017a.

Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 357–368. PMLR, 2017b.

Pete Florence, Corey Lynch, Andy Zeng, Oscar A. Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 158–168. PMLR, 2022.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang. A tutorial on energy-based learning. In Gökhan Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan, editors, *Predicting Structured Data*. MIT Press, 2006.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500, 2023. doi: 10.1109/ICRA48891.2023.10160591.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. doi: 10.1109/CVPR.2015.7298965.

Sriraam Natarajan, Saket Joshi, Prasad Tadepalli, Kristian Kersting, and Jude Shavlik. Imitation learning in relational domains: A functional-gradient boosting approach. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 1414–1420, 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-239.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, Anikait Singh, et al. Open x-embodiment: Robotic learning datasets and rt-x models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903, 2024. doi: 10.1109/ICRA57147.2024.10611477.

OpenAI. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/, April 2025. Accessed: 2026-05-12.

Jyothish Pari, Nur Muhammad Mahi Shafiullah, Sridhar Arunachalam, and Lerrel Pinto. The surprising effectiveness of representation learning for visual imitation. In *Robotics: Science and Systems*, 2022.

Noah Patton, Kia Rahmani, Meghana Missula, Joydeep Biswas, and Isil Dillig. Programming-by-demonstration for long-horizon robot tasks. *arXiv preprint arXiv:2305.03129*, 2023.

Dean A. Pomerleau. ALVINN: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, volume 1, pages 305–313, 1988.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635. PMLR, 2011.

Tom Silver, Kelsey R. Allen, Alex K. Lew, Leslie Pack Kaelbling, and Josh Tenenbaum. Few-shot Bayesian imitation learning with logical program policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10251–10258, 2020. doi: 10.1609/aaai.v34i06.6587.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5052–5061. PMLR, 2018.

Jimmy Xin, Linus Zheng, Kia Rahmani, Jiayi Wei, Jarrett Holtz, Isil Dillig, and Joydeep Biswas. Programmatic imitation learning from unlabeled and noisy demonstrations. *arXiv preprint arXiv:2303.01440*, 2023.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.016.

6 Appendix

A Bayesian Logical Policy Learning Details

This appendix gives the details of the Bayesian logical policy-learning component used in TwoLIP.

A.1 Posterior over structured classifiers

Given a feature library $\Phi = \{\phi_1, \ldots, \phi_m\}$, each state–action pair (s, a) is represented by $x_\Phi(s, a) \in \{0, 1\}^m$. Let $\mathcal{H}_\Phi$ denote the class of structured classifiers over this representation. Each classifier $h \in \mathcal{H}_\Phi$ predicts whether an action should be treated as admissible in a state, with $h(x_\Phi(s, a)) = 1$ indicating admissibility and $h(x_\Phi(s, a)) = 0$ indicating inadmissibility.

We define a posterior over classifiers,

$$p(h \mid \mathcal{D}_{\text{sa}}, \Phi) \propto p(\mathcal{D}_{\text{sa}} \mid h, \Phi)\, p(h \mid \Phi),$$

where $\mathcal{D}_{\text{sa}} = \{(s_i, a_i, y_i)\}_{i=1}^n$ is the contrastive state–action dataset. Positive examples correspond to demonstrated actions, $y_i = 1$, while negative examples correspond to alternative actions, $y_i = 0$. The likelihood rewards classifiers that agree with these labels, while the prior favors simpler logical policies.

A.2 Soft likelihood with allow/reject penalties

For each labeled example (s_i, a_i, y_i), let $\hat{y}_i = h(x_\Phi(s_i, a_i))$. A deterministic classifier would either agree or disagree with the label. However, because the demonstrations are few and the negative set may contain actions that are not necessarily impossible, we use a softened likelihood that penalizes different types of mistakes rather than assigning zero probability to any imperfect policy.

One convenient form is an exponential penalty model:

$$p(\mathcal{D}_{\text{sa}} \mid h, \Phi) \propto \exp\left(-\sum_{i=1}^{n} \omega_i \left[\lambda_{\text{fn}} \mathbf{1}\{y_i = 1, \hat{y}_i = 0\} + \lambda_{\text{fp}} \mathbf{1}\{y_i = 0, \hat{y}_i = 1\}\right]\right).$$

Here, ω_i is an optional example weight, λ_{fn} penalizes rejecting a demonstrated action, and λ_{fp} penalizes allowing a negative action. In our setting, false negatives are usually more severe: if h rejects the demonstrated action, then the induced policy cannot reproduce the expert choice in that state. False positives are also penalized, but they may be treated more softly because some alternative actions can be behaviorally plausible, especially when only a few demonstrations are available.

Equivalently, this likelihood can be written as a product of per-example probabilities,

$$p(\mathcal{D}_{\text{sa}} \mid h, \Phi) = \prod_{i=1}^{n} p(y_i \mid h, x_\Phi(s_i, a_i))^{\omega_i},$$

where

$$p(y_i = 1 \mid h, x_i) = \begin{cases} 1 - \epsilon_{\text{fn}}, & h(x_i) = 1, \\ \epsilon_{\text{fn}}, & h(x_i) = 0, \end{cases} \qquad p(y_i = 0 \mid h, x_i) = \begin{cases} 1 - \epsilon_{\text{fp}}, & h(x_i) = 0, \\ \epsilon_{\text{fp}}, & h(x_i) = 1. \end{cases}$$

The parameters ϵ_{fn} and ϵ_{fp} are the softened error probabilities corresponding to the false-negative and false-positive penalties. This form avoids making the posterior collapse when no candidate classifier perfectly separates the contrastive dataset.

A.3 Structural prior

The prior $p(h \mid \Phi)$ encodes a preference for compact logical explanations. Since each classifier is represented as a logical rule over Boolean features, we penalize structural complexity, such as the number of clauses, the number of feature literals, or the depth of the corresponding decision tree. A generic form is

$$p(h \mid \Phi) \propto \exp(-\alpha \, \mathrm{complexity}(h)),$$

where $\alpha \geq 0$ controls the strength of the simplicity bias. This prior discourages policies that fit the demonstrations using overly specific conjunctions of features, which are more likely to encode accidental correlations in the small demonstration set.

In our implementation, the structured classifiers are instantiated using decision trees over the induced Boolean features. A path to a positive leaf corresponds to a conjunction of feature literals, and the set of positive leaves forms a disjunction. Thus, decision-tree complexity provides a natural proxy for logical policy complexity.

A.4 Approximate posterior inference

Exact inference over $\mathcal{H}_\Phi$ is intractable because the number of logical formulas grows combinatorially with the number of features. We therefore approximate the posterior with a weighted set of particles,

$$q(h) = \sum_{k=1}^{K} w_k \delta_{h_k}(h),$$

where each h_k is a candidate structured classifier and w_k is its normalized posterior weight.

Candidate particles are generated by repeatedly training logical learners on subsets of the induced feature library. For a feature subset $\Phi_\ell \subseteq \Phi$, we construct the corresponding feature matrix

$$X_\ell = \{x_{\Phi_\ell}(s_i, a_i) : (s_i, a_i, y_i) \in \mathcal{D}_{\mathrm{sa}}\}.$$

A decision-tree learner is then fit to (X_ℓ, y), producing one or more candidate classifiers. Each candidate classifier is scored using the posterior objective,

$$\tilde{w}_k = p(\mathcal{D}_{\mathrm{sa}} \mid h_k, \Phi)\, p(h_k \mid \Phi),$$

and the weights are normalized as

$$w_k = \frac{\tilde{w}_k}{\sum_{j=1}^{K} \tilde{w}_j}.$$

This particle approximation allows TwoLIPto maintain multiple plausible policies instead of committing to a single explanation of the demonstrations.

A.5 Posterior mixture policy

Each classifier h_k induces an admissibility prediction for every candidate action. The posterior mixture combines these predictions into a soft admissibility score,

$$S_q(s, a) = \mathbb{E}_{h \sim q}[h(x_\Phi(s, a))] = \sum_{k=1}^{K} w_k h_k(x_\Phi(s, a)).$$

At test time, the policy selects an action with maximum posterior expected admissibility,

$$\pi_q(s) \in \arg\max_{a \in \mathcal{A}} S_q(s, a).$$

This mixture can be interpreted as a soft state–action scoring model: actions marked admissible by many high-posterior policies receive higher scores, while actions rejected by most policies receive lower scores. When the few demonstrations underdetermine the correct policy, this mixture reduces sensitivity to any single classifier and allows several plausible explanations to contribute to action selection.

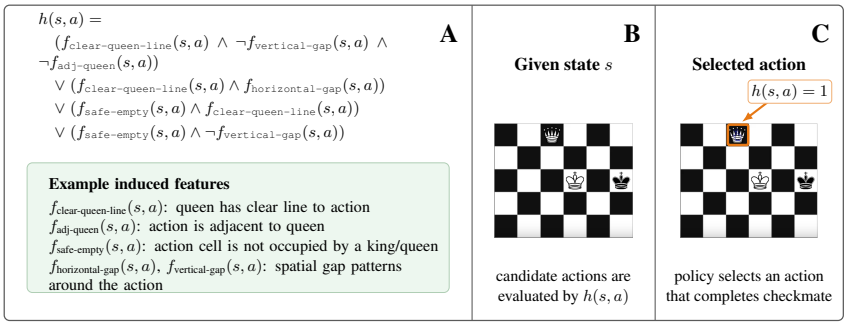


Figure A.1: Example learned policy for Checkmate Tactic. (A) The learned logical policy composes LLM-induced features into a readable rule over state–action pairs. (B) Given a board state, candidate actions are evaluated by the policy. (C) The selected action satisfies the learned rule and solves the instance.

B Proof of Proposition 1

Proof. Fix a feature library Φ. The representation x_Φ partitions $\mathcal{D}_{\text{sa}}$ into feature buckets, where two examples belong to the same bucket if they have the same feature vector. Let $\mathcal{B}_\Phi$ denote the set of buckets induced by Φ. For each bucket $B \in \mathcal{B}_\Phi$, define

$$n_B^+ = |\{(s,a,y) \in B : y = 1\}|, \qquad n_B^- = |\{(s,a,y) \in B : y = 0\}|.$$

Any deterministic classifier whose prediction depends only on $x_\Phi(s,a)$ must assign the same label to all examples in a bucket B. Therefore, the best possible prediction on B is the majority label, and the minimum number of mistakes on that bucket is

$$\min(n_B^+, n_B^-).$$

Thus,

$$R^*(\Phi) = \frac{1}{|\mathcal{D}_{\text{sa}}|} \sum_{B \in \mathcal{B}_\Phi} \min(n_B^+, n_B^-).$$

Now consider the augmented feature library $\Phi \cup \Psi$. Adding features can only refine the original partition: each bucket $B \in \mathcal{B}_\Phi$ is split into sub-buckets $B_1, \ldots, B_m$ under the representation $x_{\Phi\cup\Psi}$. For these sub-buckets,

$$\sum_{j=1}^m n_{B_j}^+ = n_B^+, \qquad \sum_{j=1}^m n_{B_j}^- = n_B^-.$$

The contribution of the refined buckets to the optimal empirical error is

$$\sum_{j=1}^m \min(n_{B_j}^+, n_{B_j}^-).$$

For nonnegative numbers a_j, b_j, we have

$$\sum_{j=1}^m \min(a_j, b_j) \le \min\left(\sum_{j=1}^m a_j, \sum_{j=1}^m b_j\right).$$

Applying this with $a_j = n_{B_j}^+$ and $b_j = n_{B_j}^-$ gives

$$\sum_{j=1}^m \min(n_{B_j}^+, n_{B_j}^-) \le \min(n_B^+, n_B^-).$$

Therefore, each original bucket contributes no more error after adding Ψ. Summing over all original buckets yields

$$R^*(\Phi \cup \Psi) \leq R^*(\Phi).$$

If the repair step is accepted because it reduces the collision-induced empirical-error bound, then for at least one original bucket B,

$$\sum_{j=1}^{m} \min(n^+_{B_j}, n^-_{B_j}) < \min(n^+_B, n^-_B),$$

while all other buckets contribute no more error than before. Hence the total optimal empirical error strictly decreases:

$$R^*(\Phi \cup \Psi) < R^*(\Phi).$$

□

C Additional Experimental Results

This appendix includes additional plots complementing the main experiments, including the training success curves corresponding to the held-out test results reported in the main text.

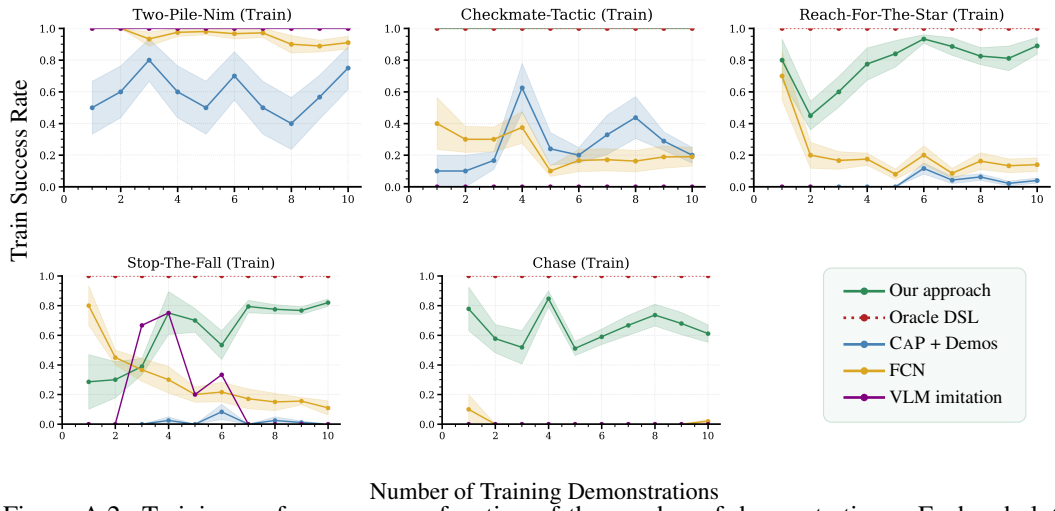
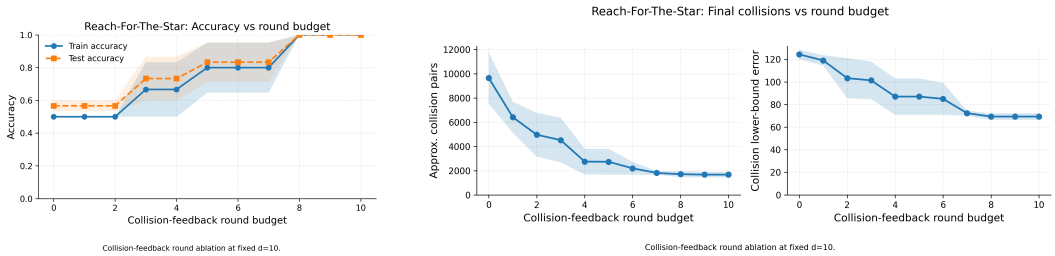


Figure A.2: Training performance as a function of the number of demonstrations. Each subplot reports mean task-completion success on training task instances over 10 random seeds.

D Prompt Templates

We show simplified prompt templates used by TwoLIP. The LLM is used for initial feature-template generation and collision-guided feature repair, rather than for directly producing actions or policies. Task-specific demonstrations, collision examples, and implementation details are replaced by placeholders.



(a) Accuracy vs. repair rounds

(b) Collision metrics vs. repair rounds

Figure A.3: Effect of collision-guided repair on *RFTS* with $d = 10$.

D.1 Feature-Template Generation Prompt

Feature-Template Generation Prompt

SYSTEM
You are an expert feature-library designer for program synthesis and Logical Program Policies in grid-based games.
Your task is to read expert demonstrations (s, a, s') and propose a compact, expressive library of Boolean feature templates $f_i(s, a)$.
The goal is not to enumerate many variants, but to introduce the minimal set of structurally distinct predicate families necessary to express the expert's decision rule.

ENVIRONMENT
{description of the grid state, clicked-cell action, transition dynamics, and feature-function interface}

TEMPLATE CONSTRAINTS
{allowed placeholders, token-role rules, action validation logic, and examples of forbidden patterns such as hard-coded tokens or duplicate complements}

DESIGN PRINCIPLES
{features should be structurally expressive, minimal, general, scale-invariant, action-grounded, and non-redundant}

INPUT: DEMONSTRATIONS
{expert trajectories summarized with board structure, local views, ray features, distances, and transition effects}

TASK
Infer the structural distinctions that separate expert actions from plausible alternatives, generate feature templates that encode those distinctions, and stop once semantic coverage is sufficient.

OUTPUT FORMAT
Return only valid JSON:

```
{
  "features": [
    {"id": "...", "name": "...", "source": "..."}
  ]
}
```

D.2 Collision-Repair Prompt

Collision-Repair Prompt

SYSTEM
You are an expert feature-library repair agent for Logical Program Policies in grid-based games.
Your task is to propose new Boolean feature functions $f_i(s, a)$ that repair feature collisions.
A collision occurs when positive and negative examples have identical feature vectors under the current feature library, but different labels.

OBJECTIVE
Introduce genuinely new structural distinctions that separate the colliding examples and generalize beyond the provided collision bucket.

ENVIRONMENT
{description of the grid state, clicked-cell action, allowed token constants, and feature-function interface}

IMPLEMENTATION CONSTRAINTS
{allowed tokens, action validation logic, no imports, no raw strings, no unsupported helpers}

ANTI-OVERFITTING CONSTRAINTS
{no hard-coded coordinates, no board-size assumptions, no dataset-specific thresholds, no scan-order anchors, no memorization}

PREFERRED STRUCTURAL FEATURES
{ray visibility, proximity, alignment, reachability, boundary relations, gaps, between-ness, or other structural relations}

COLLISION EVIDENCE
{one or more collision buckets containing positive examples, negative examples, local/ray/global summaries, and difference hints}

BUCKET HANDLING
Analyze each collision bucket independently. If multiple buckets reveal the same missing structural distinction, propose a feature that resolves them jointly.

TASK
Compare positive and negative examples, identify missing structural invariants, and propose new non-redundant features that break the collision while generalizing to new boards and placements.

OUTPUT FORMAT
Return only valid JSON:

```
{
  "features": [
    {
      "id": "...",
      "name": "...",
      "description": "...",
      "source": "..."
    }
  ]
}
```

E Discussion/Future Work

This work focuses on the role of inductive bias in few-demonstration IL. Our goal is not to argue for a particular optimization algorithm, but to show that, in settings where demonstrations are limited, it can be useful to first learn a task-relevant feature representation and then learn a policy over that representation. This gives the learner a more structured hypothesis space than end-to-end action prediction, while still avoiding the need for a manually designed feature set. At the same time, this creates an important tradeoff: the feature-generation process must be expressive enough to make the

task learnable, but constrained enough to avoid simply memorizing the demonstrations. We include additional practical details on how we balance this tradeoff in Appendix F.

A common challenge in IL is compounding error: small mistakes made by the learned policy can move the agent into states that were not covered by the expert demonstrations, where further mistakes become more likely (Ross et al., 2011). This was not a major issue in the grid-game domains we study here, since the policy chooses among discrete candidate actions and the action space is relatively controlled. However, we expect this issue to become more important in future continuous-action settings. In continuous control, small action errors can accumulate over time and push the agent farther from the demonstrated trajectories. Extending TwoLIPto these settings may therefore require interactive data collection, recovery demonstrations, or planners that can help correct local mistakes.

A second direction is to move beyond generating task-specific features from scratch. In this work, the LLM proposes a feature library for each task, and we then repair that library using collision feedback. A natural next step is to treat the feature library itself as a reusable object. Features learned for one task could initialize the library for a related task, and new demonstrations could be used to add, revise, or remove features over time. This suggests a more lifelong or continual-learning version of TwoLIP, where the learner gradually builds a library of reusable abstractions rather than restarting feature induction for every new task.

Another interesting direction is to use the learned policies as components inside a planner. The policies learned by TwoLIPare not just black-box action predictors; they define interpretable conditions under which actions are admissible. This makes them potentially useful as skills, action filters, or proposal mechanisms for planning. For example, a planner could use a learned policy to restrict the action space to semantically plausible actions, while still searching over longer-horizon consequences. This could combine the generalization benefits of structured imitation with the robustness of planning.

Finally, our results suggest that LLMs may be especially useful not as direct policy generators, but as tools for proposing and revising abstractions. Instead of asking the model to solve the entire control problem end-to-end, TwoLIPasks it to generate candidate features, tests those features against demonstrations, and uses failures as feedback for repair. This separation gives the learning algorithm a way to inspect, filter, and improve the representation before committing to a policy. We believe this is an important direction for using language models in IL: not as standalone agents, but as sources of structured hypotheses that can be grounded and tested through interaction with data.

F Expressiveness vs. Generalization

The proposed feature-generation pipeline introduces a tradeoff between *expressiveness* and *generalization*. These two forces correspond to different goals in the learning process.

Expressiveness. Feature collisions indicate that the current representation is insufficient. Recall that a collision occurs when two state–action pairs have the same feature vector but different labels:

$$x_\Phi(s,a) = x_\Phi(s',a') \quad \text{but} \quad y \neq y'.$$

In this case, no deterministic classifier

$$h : \{0,1\}^{|\Phi|} \to \{0,1\}$$

can perfectly fit the data. Collision detection therefore identifies violations of the condition

$$\exists h \in \mathcal{H}_\Phi \quad \text{such that} \quad h(x_\Phi(s_i,a_i)) = y_i \quad \forall i.$$

To resolve collisions, the system generates additional features. If Ψ is a set of added features, then collision repair expands the effective hypothesis class:

$$\mathcal{H}_\Phi \subseteq \mathcal{H}_{\Phi\cup\Psi}.$$

Thus, collision-driven feature generation improves learnability: once collisions are removed, the training data can in principle be separated by some classifier in the expanded feature space.

Generalization. However, eliminating collisions does not guarantee generalization. If the feature map becomes too expressive, the hypothesis class can memorize the training set. For example, writing $x_i = x_\Phi(s_i, a_i)$, once every positive example has a unique feature vector, one can construct the DNF

$$h(x) = \bigvee_{i:y_i=1} \left(\bigwedge_{j:x_{ij}=1} x_j \wedge \bigwedge_{j:x_{ij}=0} \neg x_j \right),$$

which fits each positive training example individually. In this case, the empirical risk may go to zero,

$$\hat{R}_{\text{train}}(h) \to 0,$$

while the true risk may still be high:

$$R(h) = \mathbb{E}_{(s,a,y)\sim\mathcal{D}_{\text{test}}} \left[\mathbf{1}\{h(x_\Phi(s,a)) \neq y\} \right].$$

This can happen when newly generated features capture false shortcut correlations that are predictive in the demonstrations but unstable across test configurations.

Controlling the tradeoff. A natural direction for future work is to make this expressiveness–generalization tradeoff explicit in the learner. One approach is to detect and reject false shortcut features, such as predicates that rely on overly specific coordinates, object counts, or demonstration-specific artifacts. Another is to assign complexity costs to features and use these costs as part of the structural prior, biasing inference toward more stable and reusable predicates.

A second direction is to filter features across demonstrations. After generating a feature library, one can score features by how consistently they are useful across different demonstrations, and remove features that only activate for a small number of examples. This would discourage features that explain isolated demonstrations but do not capture the broader task structure.

Finally, the hypothesis class itself can be restricted to reduce memorization. For decision-tree learners, this can be done through standard capacity controls such as maximum depth, minimum leaf size, or cost-complexity pruning. Similarly, collision repair can be stopped early instead of continuing until all training collisions are removed, since fully eliminating collisions may over-specialize the feature space. Together, these mechanisms provide ways to balance the need for expressive features with the need for policies that generalize.

G Task Details

All tasks use the Generalization Grid Games interface: each state is an $h \times w$ grid with discrete object types, and each action selects a grid cell $a \in \{1, \ldots, h\} \times \{1, \ldots, w\}$. The transition is deterministic, and an episode is successful if the task goal is reached within the horizon.

TwoPileNim. *TwoPileNim* is a spatial variant of Nim. The grid contains token cells arranged into two piles, and an action removes tokens according to the game dynamics. The goal is to select moves that follow the expert Nim strategy and reach a winning terminal configuration.

CheckmateTactic. *CheckmateTactic* is a chess-tactics task represented on a grid. The state contains chess pieces, and the agent must select the move or target cell that progresses toward a checkmating configuration. Success requires recognizing the relevant spatial relations between pieces, such as attacks, alignment, and king safety.

StopTheFall. *StopTheFall* contains gravity-like dynamics in which objects may fall unless the agent intervenes. The goal is to click cells that prevent the falling object from reaching an unsafe configuration, for example by placing or selecting supporting cells. The task requires reasoning about vertical structure, support, and future motion.

ReachForTheStar. *ReachForTheStar* is a grid-construction/navigation task in which the agent must reach a star. Actions modify the grid so that the agent can make progress toward the star, often requiring intermediate structures rather than simply selecting the goal cell. The task tests whether the learner can infer reusable spatial concepts such as support, adjacency, connectivity, and progress toward the target.

Chase. *Chase* requires the agent to catch or reach a moving target while respecting the grid layout. The expert behavior often involves intermediate control decisions before directly pursuing the target. This makes the task challenging because successful behavior depends not only on distance to the target, but also on the semantics of action cells, obstacles, and the current stage of the strategy.

Reach-The-Corner. *Reach-The-Corner* (RTC) is a new task introduced for the transfer experiment. It uses the same environment family as *ReachForTheStar*, but changes the goal from reaching a star to reaching a designated corner of the grid. This preserves much of the underlying environment structure while changing the task objective, making it useful for testing whether features induced on *ReachForTheStar* transfer to a related task.