Does Asking for Help Mitigate Goal Misgeneralization?

Anonymous authors
Paper under double-blind review

Abstract

Reinforcement learning (RL) agents have achieved impressive results but can struggle to generalize, leading to unexpected and potentially unsafe behavior. Prior work discovered that, under certain distribution shifts in the test environment, agents competently pursue incorrect goals, even when they behave near-optimally in the training environment. This phenomenon, termed *goal misgeneralization,* occurs when RL agents find proxy goals that correlate with the intended goal during training but diverge at test time. We investigate the extent to which goal misgeneralization can be mitigated by allowing agents to ask an expert for help. We find that, in existing goal misgeneralization environments, simple uncertainty estimation methods and heuristics can mitigate apparent GMG failures. However, this success relies on a particular property of existing benchmarks: achieving the proxy goal has no negative consequences, making delayed failure detection viable. We propose modified versions of existing goal misgeneralization benchmarks that do not allow recovery. We find that this drastically reduces the ability of existing query strategies to mitigate goal misgeneralization, highlighting their limitations. We encourage future work to study this more challenging class of goal misgeneralization failures with stricter failure modes. Code and videos are available at `https://asking-for-help-gmg.github.io/`

1 Introduction

Much prior work has focused on finding optimization objectives that make reinforcement learning (RL) systems useful and safe to use (Kaufmann et al. 2025; Ji et al. 2025). For example, an important aspect of training state-of-the-art large language models is learning a reward function from humans (Christiano et al. 2017; Ziegler et al. 2019; Stiennon et al. 2020; Ouyang et al. 2022) or optimizing verifiable rewards (Guo et al. 2025). However, machine learning systems are generally unreliable in the presence of distribution shifts (Quiñonero-Candela et al. 2008). In RL, prior work has shown that agents can find proxy goals that yield high performance at training time but cause them to capably pursue incorrect goals and fail at the intended objective when the distribution shifts (Langosco et al. 2022; Shah et al. 2022). This phenomenon, termed *goal misgeneralization* (GMG), could lead to particularly dangerous outcomes if powerful RL systems capable of affecting the real world pursue unintended objectives. These failure modes are challenging to detect because the systems appear to be well-behaved and pursue the correct goal during training.

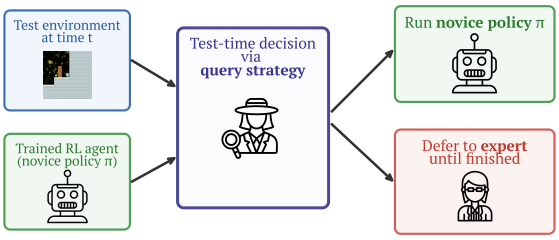


Figure 1: **Overview of the asking-for-help framework.** A query strategy decides whether to continue running the novice or to defer to the expert for the rest of the interaction.

As a real-world example, Wang et al. (2025) observed potential GMG behavior in state-of-the-art robotics systems. The large-scale robotics model π_0 (Black et al. 2025) can solve a variety of tasks when given natural-language commands. However, the training setup did not account for all situations that might be encountered during deployment. Wang et al. found that if given nonsensical instructions, the π_0 policy has a strong bias toward pursuing default objectives, such as picking up objects that appear frequently in the

training data. Although these failure modes are harmless, capable models with a strong bias towards action could damage hardware or harm humans. In such cases, it may be safer for the system to recognize that the situation is beyond its training data and defer rather than take an incorrect action.

Because the distribution shift occurs after training, GMG is a natural candidate for deployment-time oversight rather than training-time correction. Additionally, the deployed policy might be frozen: retraining may be expensive, the policy may be part of a larger pre-trained system, or modifying it after deployment may be difficult for safety or regulatory reasons. Though AI training environments are continually closing the gap toward the deployment setting, anticipating all possible deployment states might be intractable. Reasons for this include the inherent gap in simulation fidelity, biases introduced by data collection, and the non-stationarity of the real world—including the fact that the act of deploying an agent itself changes the environment. A last resort safety mechanism, however, does not need to solve the task itself. Instead, it may be enough for the system to notice when autonomous control becomes risky and hand control over to a more reliable expert. This motivates an asking-for-help framework (Scerri et al. 2002; Reddy et al. 2018; Nguyen and Daumé III 2019): a trained novice policy acts by default, but a query strategy can decide when to defer to an expert. A useful query strategy must ask for help before the novice's proxy-goal pursuit causes harm, while also avoiding unnecessary queries in situations where the novice would have succeeded on its own.

Many highly autonomous systems deployed in the real world already rely on some form of human oversight or expert fallback, for example, by relying on human supervision or the ability to ask remote human operators for help in situations the system cannot reliably resolve by itself (Tesla, Inc. 2025; Waymo LLC 2024). In this work, we evaluate the feasibility of incorporating an asking-for-help framework to mitigate the consequences of GMG. Figure 1 illustrates this framework. To decide whether to ask the expert, we compare various established methods for detecting out-of-distribution (OOD) states and estimating uncertainty (Lakshminarayanan et al. 2017; Ruff et al. 2018). We find that the success of these methods makes use of a specific property of the environment: following the proxy goal has no irreversible side effects, so the novice can commit to the wrong goal and still recover afterward. This lets a delayed failure detector intervene after the fact. In more safety-critical settings, the proxy goal may be irreversible—a robot may damage an object, consume a resource, or enter an unsafe region—precluding successful completion before any delayed detector can trigger. Our main contributions are as follows:

1. We show that expert deferral can mitigate GMG in standard benchmarks, but that this success largely depends on recoverability: pursuing the proxy goal has no negative consequences.
2. We introduce irrecoverable variants of existing GMG benchmarks, in which pursuing the proxy goal terminates the episode without reward.
3. We find that methods that perform well on standard benchmarks largely fail when deferral is required before the proxy-goal is achieved.

2 Related Work

Goal misgeneralization. Prior work coined the term goal misgeneralization (GMG), demonstrating that RL agents can retain their capabilities under distributional shifts while pursuing unintended proxy goals (Langosco et al. 2022; Shah et al. 2022). Langosco et al. introduced three modified Procgen (Cobbe et al. 2020) environments, Coinrun, Maze, and KeysAndChests, which induce GMG by changing the distribution of goal objects between training and test. These environments have become the primary benchmark for empirical work on GMG mitigation, with subsequent work evaluating concept extrapolation in Coinrun (Armstrong et al. 2023), language-model-guided reward learning in Maze (Barj and Sautory 2024), and custom grid-worlds with the same proxy-goal structure (Abdel Sadek et al. 2025). Shah et al. (2022) demonstrate GMG in a broader set of domains, including grid-worlds with resource depletion and agent-induced distribution shifts. However, these settings were introduced to establish the breadth of GMG as a phenomenon, and have not been adopted as benchmarks for evaluating mitigation methods.

A property shared by the Procgen environments that serve as the standard testbed is that pursuing the proxy goal carries no negative consequences: the agent can follow the proxy goal and still recover afterward. Some

of Shah et al.'s environments, particularly Tree Gridworld, where overchopping depletes a shared resource, do not share this property, but have not been used to study mitigation. Our irrecoverable environment variants (Section 5.4) address this gap for the benchmarks most commonly used in GMG mitigation works.

Armstrong et al. (2023) propose a mitigation strategy that involves continuing to learn in the deployment environment, albeit without receiving a reward signal at deployment time. Abdel Sadek et al. (2025) theoretically prove a mitigation strategy in a relaxed setting, where OOD levels naturally occur with small, non-zero probability at training time. Barj and Sautory (2024) make use of the priors learned by large language models, to aid in GMG mitigation. Conversely, in our problem setting, we assume a frozen training setup that precludes changes, we assume no knowledge of OOD environments at training time, and study the extent to which mitigation is possible without any learned human priors. This is motivated by current practice using pre-trained systems and compatibility with systems that must be frozen for safety or regulatory reasons, and sidesteps potential safety concerns when integrating learned human priors.

Shortcut learning. Despite impressive results in many domains, a common failure mode of deep learning systems is *shortcut learning* (Geirhos et al. 2020; Ye et al. 2026). Shortcut learning describes the tendency of neural networks to learn brittle, non-semantic decision rules. This causes deep learning systems to learn just enough to achieve good results on benchmarks, only to fail after deployment. GMG can be framed as a shortcut-learning failure of particular interest because it occurs even when objectives are correctly specified. For example, in Langosco et al. (2022)'s GMG environment, the reward function in the training environment is identical to that in the test environment. While various mitigation strategies to shortcut learning have been studied in prior work (Ye et al. 2026; Pathak et al. 2026), we focus on ceding control to an expert, which we believe is a promising approach to addressing GMG.

Interactive imitation learning. *Interactive imitation learning* (interactive IL; Kelly et al. 2019) is a related problem setting to GMG. Robot-gated interactive IL methods, such as ThriftyDAgger (Hoque et al. 2022), study various ways of deciding when to query an expert to help with learning from demonstrations. Our setting shares the core challenge of identifying states where the autonomous policy is unreliable. However, we evaluate methods that are agnostic to how the novice agent is trained. This allows our method to be used with large pre-trained models, which might be intractable to train with a human in the loop. We use the supervision signal at deployment time, not to increase the capabilities of our methods, but to improve their safety and reliability.

Asking for help. Allowing control of an autonomous system to be deferred to a human expert can improve the system's reliability. Most prior work in RL that studies this framework, which we refer to as *asking for help*, aims to improve reliability at training time and for safe exploration (Clouse 1996; Garcia and Fernandez 2012; Garcia and Fernandez 2015). Similarly, human-in-the-loop ML integrates human expertise into various stages of model training and inference (Wu et al. 2022; Mosqueira-Rey et al. 2023; Retzlaff et al. 2024). In our work, we focus on solutions that defer to human guidance at *deployment time* to study a setting that we deem relevant for current state-of-the-art methods. Though prior work exists on asking for help at deployment time for decision-making systems (Scerri et al. 2002; Reddy et al. 2018; Nguyen and Daumé III 2019), we specifically study whether this helps tackle GMG.

Danesh et al. (2025) introduce YRC-0, a setting in which an agent must learn when to yield to an expert under novel test conditions without interacting with the expert during training. Our work builds on the YRC codebase and adapts a similar coordination framework to goal-misgeneralization environments.

Query strategies. The query strategies we evaluate are drawn from common families of OOD detection and uncertainty estimation. Following standard OOD taxonomies, these include confidence/output-based scores, distance-based or one-class detectors, and ensemble-based uncertainty estimates. MaxProb and MaxLogit are simple output-based confidence scores based on maximum softmax probability and maximum logit, respectively (Hendrycks and Gimpel 2017; Hendrycks et al. 2022). DeepSVDD uses learning-based one-class detection on raw observations (Ruff et al. 2018), while Ensemble uses deep-ensemble disagreement as an uncertainty signal (Lakshminarayanan et al. 2017). These choices overlap with the signals used in closely related deployment-time deferral work: ThriftyDAgger gates interventions using ensemble-based novelty and learned risk (Hoque et al. 2022), and YRC-Bench evaluates several thresholded confidence and OOD scores for yielding and requesting control (Danesh et al. 2025). Thus, our methods are intended as representative

simple baselines for deciding when a frozen autonomous policy should defer, rather than as an exhaustive comparison of OOD-detection methods.

3 Problem Setting

We assume the objective is to create a semi-autonomous system that successfully performs a given task with access to expert supervision.

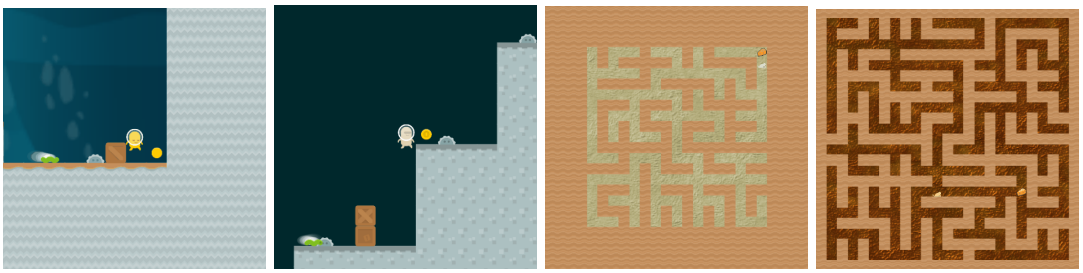
Markov decision processes. A *task* is represented by a Markov decision process (MDP). We define an undiscounted episodic MDP $M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \rho_0)$, where $\mathcal{S}$ is the set of possible states of the environment (including a set of terminal states $\mathcal{S}_{\text{term}} \subseteq \mathcal{S}$), $\mathcal{A}$ the set of all possible actions, $\mathcal{T}$ defines the transition dynamics of the environment, $\mathcal{R}$ the scalar reward function, and ρ_0 is the distribution over initial states. The goal in RL is to find the optimal policy $\pi^*(s)$ that maximizes the reward when rolled out from any of the starting states. Evaluation is done in *episodes*, which start with a state drawn from ρ_0 and roll out the policy until a terminal state is reached, truncating after a maximum number of steps. The sum of rewards received across the episode's transitions is called *return*. Some environments in our experiments are partially observable, so the policy inputs are observations $o = \mathcal{O}(s)$ derived from the state by a deterministic mapping.

What is goal misgeneralization? The tasks we study in our setting are selected to induce GMG. We use the observational definition of GMG introduced by Langosco et al. (2022). An RL policy, which we call the novice, is trained to maximize the reward $\mathcal{R}$, the **intended objective**, in a "training" distribution of starting states. We call this distribution ρ_0^{id}, and refer to the states as ID starting states. Though trained on $\mathcal{R}$, the novice instead learns to maximize a **proxy objective** $\mathcal{R}'$ because $\mathcal{R}'$ perfectly correlates with $\mathcal{R}$ in the training distribution. GMG occurs when there is a shift in the distribution of starting states at deployment/"testing" time, and following the learned behavior that maximizes $\mathcal{R}'$ leads to a low reward relative to the intended objective $\mathcal{R}$. We refer to the shifted distribution of starting states, ρ_0^{ood}, as OOD. In our setting, $\text{supp}(\rho_0^{\text{id}}) \cap \text{supp}(\rho_0^{\text{ood}}) = \emptyset$. However, the states all intuitively share enough semantically meaningful structure that a policy exists that performs optimally in both ID and OOD states.

The ask for help tradeoff. We assume access to a novice agent trained only on ID starting states and susceptible to GMG. Additionally, we can query a more capable *expert agent*, such as a human providing supervision or a more capable learned model that is costly to query. The expert achieves a high return with respect to $\mathcal{R}$ both ID and OOD. We do not make assumptions about how either agent was trained, just that we can query them in arbitrary states. We assume that there is some nonzero cost to querying the expert, and we do not want the system to excessively ask for help. Thus, there is an inherent trade-off between the system's performance, which can increase when the expert is queried, and how often it asks for help.

When the agent asks for help, we cede control to the expert until the episode is finished. With the human supervisor as the expert in mind, we do this for several reasons. Firstly, we assume meaningful human feedback occurs at timescales that are longer than a single timestep, and arbitrarily alternating between the expert and the novice would be prohibitive. Similarly, at the timescale of a single task, we imagine the cost of switching to the human outweighs the cost of keeping the human in control for some potentially wasted timesteps. Additionally, we assume that when an OOD situation has been detected, a human supervisor might not trust the agent enough to give control back to the autonomous system. We leave the challenge of fine-tuning the exact timing of expert control to future work. To make the decision whether to query an expert in a given episode non-trivial, at test time, we sample from ρ_0^{id} and ρ_0^{ood} with equal probability, unlike Langosco et al. (2022), where testing is always done with ρ_0^{ood}.

Query strategies. A query strategy receives as input the current observation and the novice's internal state and decides whether to request assistance. Since the training environment is known in advance, we grant query strategies access to it. For example, some strategies utilize observations collected from the training environment during their own training. Additionally, some query strategies use the novice's internal state. Importantly, the query strategy is *not* given prior knowledge of the OOD environment or of the form the distribution shift will take—it must learn to identify unfamiliar situations from the training data alone.



(a) `Coinrun` ID (b) `Coinrun` OOD (c) `Maze` ID (d) `Maze` OOD

Figure 2: The `Coinrun` and `Maze` Procgen environments. In this paper, we study the feasibility of seeking expert assistance to mitigate GMG in the two RL environments `Coinrun` and `Maze`. The layouts of the environments are randomly generated. Both feature a goal location, demarcated by a coin and a piece of cheese, respectively. The goal location is deterministic in the in-distribution version of each environment, shown in panels a and c, and randomly placed in the out-of-distribution version, shown in panels b and d.

4 Experimental Setup

We study games that prior work has used to demonstrate GMG. We perform four independent repetitions of the same evaluation procedure, each with a different fixed seed. This results in four different experts, four results per method, etc. We report averages over seeds and the interquartile range, unless noted otherwise.

4.1 Environments

We closely follow the GMG setup of Langosco et al. (2022), which introduced three total GMG environments, a subset of the 16 Procgen games, with the other 13 not inducing any known GMG failures. We choose two out of the three games in which the authors demonstrated GMG failures, `Coinrun` and `Maze`. We leave the third game, `KeysAndChests`, which introduces a qualitatively different type of distribution shift, to future work. The environments are modified versions of Procgen (Cobbe et al. 2020). Each Procgen environment is an MDP and defines a distribution of *levels*, each of which is uniquely defined by a starting state of the MDP. Each level has a procedurally generated layout, while the rules of the game remain the same. Procedural generation controls the changing "physical" layout of the level and its visual style. `Coinrun` and `Maze`, are illustrated in Figure 2. For more details on these environments, see Appendix A.1. As explained in Section 3, the training distribution consists of only ID levels, and the testing distribution selects between ID OOD levels with equal probability.

4.2 Decision Making Agents

Our problem setting assumes access to a novice and an expert, which perform actions towards solving the task. We use PPO (Schulman et al. 2017) to train a novice on the training levels. In place of a human expert, we train PPO on the test levels as a competent deferral target. We train both novice and expert in `Coinrun` with the hyperparameters used by Langosco et al. (2022), which have been shown to achieve near-optimal results in their respective training distributions. In `Maze` we found that a naively trained expert learns brittle behavior that causes it to not reliably solve the maze after deferral. Consequently, we modify the original settings by randomly initializing starting states and a larger number of training timesteps, which leads to a robust expert. More details of this analysis can be found in Section 5.2. All hyperparameters can be found in Appendix A.2 and policy training results in Appendix A.3.

4.3 Query Strategies

In this work, we analyze the feasibility of using established OOD detection methods, uncertainty estimation techniques, and a heuristic method as query strategies. All methods output a score $g(\phi)$, which we interpret as the deferral score and whose exact calculation depends on the method. The parameter ϕ is the state of our system, containing the environment's most recent observation and the novice's internal state. Every

method is evaluated for various threshold parameters τ before evaluation, which control the frequency of asking for help by deferring to the expert when $g(\phi) > \tau$. As a reminder, when the system defers to the expert, the expert assumes control until the episode, and thus the level, ends.

Baselines & reference lines. We establish three references for interpreting the performance of the methods and a random baseline. The Novice reference illustrates the performance achieved when the novice acts autonomously without ever requesting help. The Expert reference shows the performance when the expert plays every level from the beginning. Between these, the Random baseline is the expected performance when requesting help at the start of the level with a probability proportional to τ, which produces a linear interpolation between the primary agent and expert performance as the help request rate increases.

The PartialOracle is a reference that utilizes oracle information about whether a level is ID or OOD. It is constructed to interpolate continuously across different ask-for-help budgets, while only asking for help OOD when possible. The parameter τ, which controls the ask-for-help frequency, can be any value between 0 and 2, inclusively. For $\tau \leq 1$ it asks for help in an OOD level with probability τ and in an ID level with probability 0. For $1 < \tau \leq 2$ it asks for help in all OOD levels and in ID levels with probability $\tau - 1$. We note that the PartialOracle can be outperformed when asking for help for less than 50% of the levels, by choosing *which* 50% to ask in a more beneficial way.

MaxProb & MaxLogit. These are post-hoc uncertainty prediction methods based on outputs of the novice. They require no training; they calculate $g(\phi)$ based on the novice's action distribution.

MaxProb: $g(\phi) = 1 - \max_i p_i$, where p_i are the softmax outputs of the novice. A state is seen as anomalous when all actions have similar probability, based on the intuition that the agent's action distribution should have high entropy when the agent is unsure what to do.

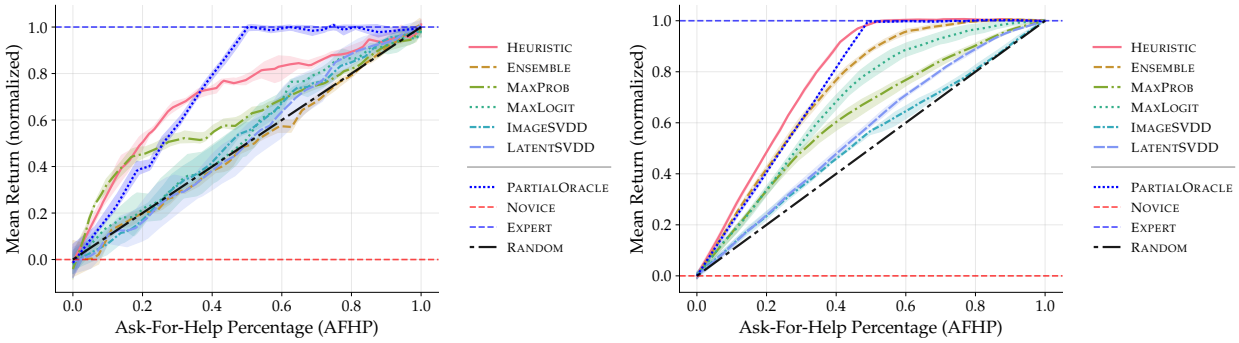
MaxLogit: $g(\phi) = -\max_i z_i$, where z_i are the logit outputs of the primary policy. Similar to MaxProb, except that we use unnormalized logits instead of probabilities.

ImageSVDD & LatentSVDD. A natural solution for detecting OOD input is to train learning-based methods. Since we assume our query strategy does not have access to OOD states during training, we focus on unsupervised methods. Selecting an appropriate method is challenging, as we assume we do not know the nature of the distribution shift in advance and because we observe that the GMG shifts we study are visually subtle. We choose DeepSVDD (Ruff et al. 2018) as a practical baseline that requires no significant modifications to our training pipeline. DeepSVDD has also been found to be a stable unsupervised method in a different setting (Han et al. 2022), performing consistently across a suite of benchmarks. We did not evaluate other learning-based OOD detection methods, and leave a detailed comparison to future work.

DeepSVDD trains a neural network for OOD detection, learning latent embeddings that minimize the radius of a hypersphere enclosing the training data embeddings. To detect OOD states, $g(\phi)$ calculates the Euclidean distance from the input observation to the center of that hypersphere. For more information on the DeepSVDD training setup, refer to Appendix A.4.

Ensemble. Another approach to OOD detection is using uncertainty estimation. We use ensembles of neural networks (Lakshminarayanan et al. 2017), an effective method that requires minimal changes to the training process. We train four additional ensemble policies in the same way as the novice, but on disjoint sets of training levels and with different seeds to control random updates. These make up the ensemble together with the primary novice used by all other methods. The score $g(\phi)$ is calculated as the variance between the action outputs of different ensemble members, averaged over the set of possible discrete actions. Note that we use the ensemble only to calculate the score $g(\phi)$. The actions performed in the environment still come from the same original novice policy that all experiments share, to provide a fair comparison.

Heuristic. A possible heuristic for asking for help is the amount of time that has passed without successfully solving the task. We implement this by having the agent randomly ask for help with constant probability p at each time step. Thus, the cumulative probability of asking for help in a given episode naturally increases as time goes on. Note that this differs from a fully random baseline that queries the expert with a fixed probability at the start of each level. While other ways to implement this heuristic are possible, we did not observe any performance difference in a small number of preliminary ablations.



(a) Experimental results for `Coinrun`.

(b) Experimental results for `Maze`.

Figure 3: **Main query strategy comparison.** We compare the return of six query strategies in `Coinrun` (a) and `Maze` (b) at test time. We compare across a range of AFHP budgets and plot the curves over medians from four independently repeated experiments, with the interquartile range drawn as the shaded region. Better methods will have higher AUCs (cf. Table 1). We find that Heuristic achieves high results across the board, MaxProb in `Coinrun`, and Ensemble in `Maze`.

4.4 Evaluation Metrics

To use an evaluation metric that trades off the cost of asking for help with the performance without assuming a specific cost, we compare methods across ask-for-help budgets. We define the ask-for-help percentage (AFHP) as the percentage of levels in which a method queried the expert, out of all evaluated levels. In our experiments, we plot a curve that has the AFHP on the x-axis and the average return in the evaluation levels on the y-axis. Better methods not only outperform others at a given budget level, but do so across a range of AFHPs, achieving a higher area under the curve (AUC).

Note that the AFHP is a derived variable we observe after evaluating a query strategy for a given threshold. The mapping from threshold to AFHP depends on the query strategy, and thresholds can vary widely. Thus, we cannot directly uniformly sample AFHP values to obtain a smooth performance curve. Instead, we implement a sampling strategy that fills uniformly arranged AFHP bins. To fill all bins, we start at extreme values for the AFHP (always ask for help, never ask for help) and use a divide-and-conquer approach to bisect the x-axis and recursively fill bins.

The structure of the environments allows the novice to correctly solve some OOD levels by accident, even when pursuing an incorrect proxy goal. To simplify comparing the query strategies, we focus quantitative results on the relative improvement over the novice, normalizing AUC values and return curves to the range defined by the mean novice and expert performance for all our experiments. That is, we normalize mean returns in `Coinrun` by $((\text{Mean expert}_{\texttt{Coinrun}}) - (\text{Mean novice}_{\texttt{Coinrun}}))$ and `Maze` respectively, so that the expert's average return is 1 and the novice's average return is 0. Expert and novice returns are read from Table 4 in Appendix A.3. We report unnormalized curves of the main results in Appendix D for reference.

5 Goal Misgeneralization With an Ask-For-Help Option

We study the feasibility of mitigating GMG by deferring to an expert. We compare six methods of asking for help in the `Coinrun` and `Maze` environments. This section is structured as follows. In Section 5.1, we find that, in existing benchmarks, asking for help can mitigate GMG failures, and learning-free methods achieve competitive performance. This success, however, comes with two important qualifications. First, it requires an expert that is robust to receiving control mid-episode, not merely competent at the task in isolation (Section 5.2). Second, the query strategies succeed largely by exploiting a property of the benchmarks themselves: the novice can pursue the proxy goal and still recover afterward (Section 5.3). In Section 5.4 we remove this property with modified environments, and find that the apparent success largely collapses.

Table 1: Area Under the Curve (AUC) for Average Return across Ask-For-Help Percentage (AFHP). IQR shows 25th–75th percentile range among four independent seeds.

(a) `Coinrun` AUC results.

Method	AUC (Median [IQR])
PartialOracle	**0.739 [0.722, 0.761]**
Heuristic	**0.701 [0.679, 0.727]**
Ensemble	0.488 [0.467, 0.511]
MaxProb	0.612 [0.596, 0.636]
MaxLogit	0.537 [0.487, 0.591]
ImageSVDD	0.526 [0.495, 0.568]
LatentSVDD	0.513 [0.444, 0.578]

(b) `Maze` AUC results.

Method	AUC (Median [IQR])
PartialOracle	0.743 [0.739, 0.748]
Heuristic	**0.787 [0.781, 0.791]**
Ensemble	0.735 [0.727, 0.744]
MaxProb	0.627 [0.607, 0.638]
MaxLogit	0.679 [0.658, 0.699]
ImageSVDD	0.532 [0.520, 0.545]
LatentSVDD	0.566 [0.558, 0.575]

5.1 Asking For Help Can Mitigate Goal Misgeneralization in Existing Benchmarks

Figures 3a and 3b report the results of comparing six methods of asking for help in the `Coinrun` and `Maze` environments, respectively. Table 1 shows the achieved AUC for the same results. We find that learning-based methods LatentSVDD and ImageSVDD have lower AUC than all other methods, except for the poorly performing Ensemble in `Coinrun`. Although DeepSVDD has proven successful at OOD detection in the computer vision domain (Ruff et al. 2018), we hypothesize that the particular form of OOD generalization failure leading to GMG in our environments is especially challenging, because the target objects make up only a few pixels of the input image and only a few frames in the overall trajectory. Similarly, LatentSVDD depends on the novice's latent features and would be impeded if the novice learns to ignore environment features relevant to OOD detection because they are not useful for achieving the proxy goal. Future work could potentially improve learning-based OOD detection methods by determining which areas and features of the observation are salient for achieving the task.

The performance of Ensemble varies highly between the two environments, achieving good performance in `Maze` but failing in `Coinrun`. One possible explanation is the entropy over the learned actions in the weak agent. Throughout the level in `Coinrun`, jumping and moving towards the right is a viable strategy to achieve the proxy goal. Thus, ensemble members might learn this behavior consistently, transferring it to OOD states with little disagreement. `Maze` requires navigating the maze by turning in various directions, and its level generation is largely uniform and symmetric across at least one axis. Consequently, when there is no optimal action—because, from the weak agent's perspective, the proxy goal has already been reached—the preferred action might be more multi-modal, causing higher variance in ensemble members.

Qualitative results in the form of sample episode rollouts can be found on our anonymized project website and the supplementary material. Anecdotally, successful episodes typically follow this pattern: the Novice follows the proxy goal, moving toward the end of the level (`Coinrun`) or the top right corner (`Maze`). After some time, the query strategy triggers, and the expert takes over, recovering the episode by backtracking to the intended goal.

Discussion. We find that DeepSVDD, the learning-based OOD detection method we study, is not well-suited for this task, performing worse than simple confidence scoring methods and simple heuristics. The image-based and latent versions may be failing for similar reasons in different ways: The visual differences between the training and test distributions are subtle, which affects both the input to the former and the latent embeddings used by the latter. These results mirror findings by Danesh et al. (2025), which found that learning-based methods were outperformed by non-learning-based methods in non-GMG decision-making settings. Surprisingly, a simple heuristic based on episode length achieves the best results in the GMG environments we study. We expand on these results in more detail in Section 5.3. These findings assume an expert robust to mid-episode handover, which we establish in Section 5.2.

5.2 Deferral Requires Experts Robust to Mid-Episode Hand-Over

The results in Section 5.1 rest on the expert being proficient not only at a task in isolation, but also at taking over partway through it. The expert must pursue the true goal from whatever state the novice has left it in, a strictly harder demand than solving the task from a standard starting state. Here, we show that this does not hold by default: in `Maze`, it requires deliberate changes to expert training. When we train an expert in `Maze` using the environment's traditional setup, we obtain an agent that achieves high return when solving levels on its own, yet yields only a small improvement over the random baseline within the asking-for-help framework (Figure 4), especially compared to the results in `Coinrun` (Figure 3a). The gap is notable because the two environments share similar GMG dynamics—the distribution of the randomly placed goal location shifts—and `Maze` lacks the "agent death" fail state. Going forward, we refer to the `Maze` expert trained in the traditional setup as the "naive expert." The naive expert no longer upper-bounds achievable performance because it underperforms the novice on ID levels (Table A.3 C), which also makes PartialOracle non-monotonic in Figure 4.[1]

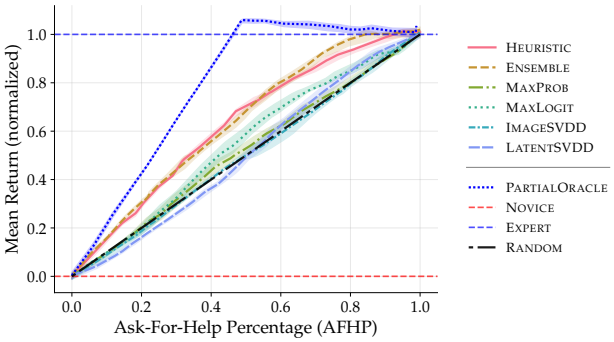


Figure 4: **Mean normalized returns in initial `Maze` experiments.** These experiments are impeded by a brittle `Maze` expert, which is why other results are performed with a more robustly-trained `Maze` expert that diverges from the traditional training setup.

To understand the low performance of query strategies with the naive expert, we performed additional analyses, which we detail in Appendix B. In summary, we observe that the performance of the naive Expert is significantly worse when it takes over from the novice than when it begins at the starting state of the same level and that this cannot be fully explained by the fact that asking for help mid-episode leaves fewer timesteps to solve the maze. Our findings suggest that naively training an Expert in `Maze` on the testing distribution leads to a brittle policy that is proficient at solving levels from the standard starting states, but whose performance deteriorates when queried mid-episode. For this reason, the `Maze` experts used outside of this section are trained to be more robust by randomly initializing starting positions (details in Appendix B.1). We find that the overall ask-for-help performance increases dramatically with this robustly trained expert.[2]

Discussion. This finding highlights that when expert deferral is an option, it is not sufficient for the expert to be proficient at solving the task in isolation; it also needs to be effective at helping less capable agents. This failure mode can also occur in the real world. For instance, a human expert might be excellent at performing a task on their own, but worse at recovering from an arbitrary task state left by the novice.

5.3 Time-Based Deferral Exploits Recoverability

In Figure 3, we observe that the Heuristic query strategy achieves high performance, leading with a large gap in `Coinrun` and even outperforming PartialOracle in `Maze`.[3] Why does this simple heuristic achieve such good performance? In both environments, pursuing the proxy goal is not mutually exclusive with the

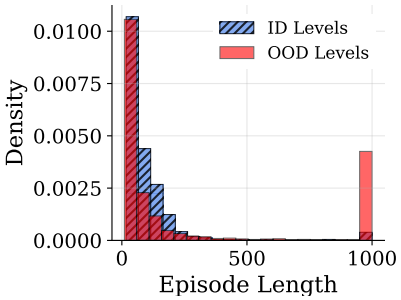
[1] In this particular setting, the Novice outperforms the Expert in ID levels, as reported in Table A.3 C. Consequently, the Expert performance is *not* an upper bound, when deferral is allowed. Instead, at an AFHP of 50, the optimal query strategy would ask for help in all OOD levels and in no ID levels, which is what PartialOracle does at that AFHP. For AFHP > 50, it is necessary to ask for help in some ID levels. Since PartialOracle does so randomly (c.f. Section 4.3), the overall mean return decreases.

[2] Note that we do not use a similar approach to attempt to train a stronger expert in `Coinrun`. The pixel-based, nearly continuous state space of `Coinrun` makes it difficult to uniformly select starting locations. Additionally, we don't observe the same degenerate behavior in `Coinrun` as we do in `Maze` when training the expert regularly.

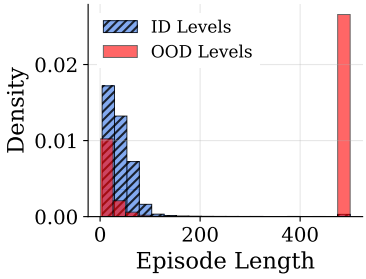
[3] As a reminder, for AFHP $\leq$ 50% PartialOracle only asks for help in OOD levels. However, which OOD levels *not* to ask for help in is randomly determined. Thus, it is possible to improve performance over this partial oracle by using the AFHP budget more efficiently and only asking for help in those levels where the novice does not succeed on accident.

subsequent recovery and fulfillment of the intended goal. Consequently, it is sufficient for methods to detect that the proxy goal does not lead to success, which makes it unnecessary to detect the distribution shift *before* the NOVICE follows the proxy goal.

We find that in this setup, the number of passed timesteps is an effective heuristic for whether to ask for help. Figure 5 shows a histogram of the episode lengths when rolling out the novice agent in ID and OOD levels. The distribution is bimodal: The higher the episode length, the lower the chance of agents making it to that length without first solving the level or dying, because solving is possible in far fewer steps than the maximum episode length. Thus, there is a peak at the beginning that quickly drops off. The second mode is at the maximum episode length, for those levels where the agent never succeeds and the episode times out. Qualitatively, the rollout videos suggest a mechanism for this timeout mode: when a GMG failure occurs, the NOVICE typically reaches the location of its proxy goal and remains stuck there until the episode ends. As a result, OOD levels have a higher concentration of maximum-length episodes than ID levels, which explains how a heuristic based on episode length can achieve a good performance-AFHP tradeoff. We perform additional experiments quantitatively analyzing the average deferral probability over time in Appendix C.



(a) `Coinrun`



(b) `Maze`

Figure 5: Distribution over episode lengths when rolling out the novice in `Coinrun` and `Maze`. We observe a bimodal distribution, with peaks for short episodes that end after fewer than 25 timesteps and for episodes of maximal length that end due to a timeout. OOD levels have more timeouts and are thus longer on average. `Coinrun` has a mean length of 99.91 ID and 265.55 OOD. `Maze` has a mean length of 41.5 ID and 338.69 OOD.

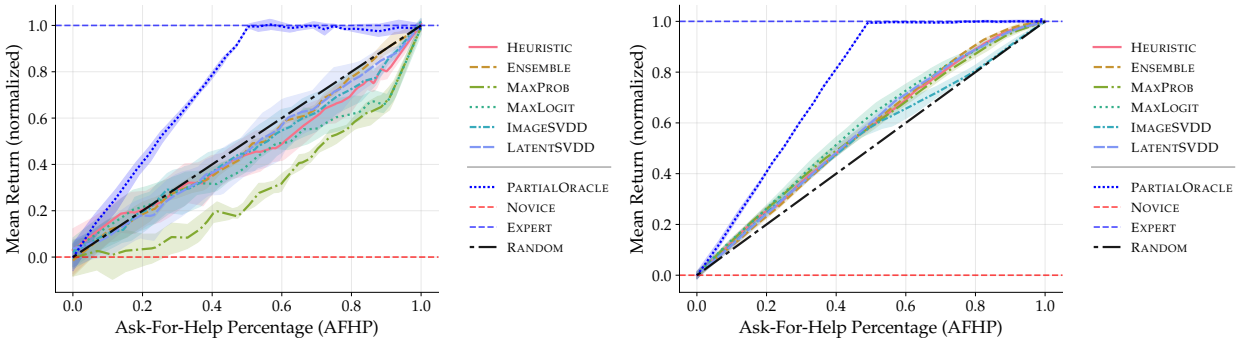
Discussion. The length heuristic works because in the existing goal misgeneralization environments following the proxy goal does not have irreversible side effects. Instead, agents can recover from pursuing the wrong goal, as long as there is enough time left to reach the true goal. In domains where proxy pursuit is non-catastrophic, a success detector combined with a timer may be a reasonable GMG mitigation: if the agent has not completed the task after long enough, defer. This may also be useful outside GMG. However, when proxy pursuit is irreversible, negative, or catastrophic, waiting to observe failure is not viable. Anecdotally, none of our query strategies—including HEURISTIC—intervene *before* the proxy goal is achieved, only detecting the failure after the fact, which motivates irrecoverable environments that we introduce in Section 5.4.

5.4 Irrecoverable Proxy Pursuit Shows the Limits of Current Query Strategies

Based on the preceding findings, we introduce modified versions of the `Coinrun` and `Maze` environments that have negative consequences for following the proxy goal. In these environments, we define reaching the proxy goal in an OOD level as reaching the location that would correspond to the goal in an ID level. When the agent reaches that location, the episode terminates with zero reward. The ID levels are unchanged. We use the same NOVICE and EXPERT from the previous experiments.

Figure 6 shows the results for these experiments in `Coinrun` and `Maze`, with the AUC values reported in Table 2. As with the previous experiments, the unnormalized figures are reported in Appendix D. In `Coinrun`, we find that none of the methods we test outperform the random baseline.[4] With ENSEMBLE being the best method and MAXLOGIT the worst, the results differ significantly from those in the recoverable setting. Since it generally takes only a fraction of the total allowed number of timesteps to achieve the proxy goal, methods that use the episode length as a heuristic are likely to ask for help too late. The low performances

[4]Note that a method will perform worse than RANDOM if it uses its AFHP budget less effectively than random selection, i.e., asks for help when there is no improvement to be gained by asking the expert, such as in ID levels.



(a) Experimental results for `Coinrun`.

(b) Experimental results for `Maze`.

Figure 6: **Results when proxy goals cause failure.** All query strategies perform worse than random in `Coinrun`. In `Maze`, the performance is drastically reduced compared to the conventional Procgen environments. Curves and shading are recorded as in Figure 3.

of Heuristic, MaxLogit, and MaxProb suggest that these methods rely on the length heuristic more so than the others. However, the low overall performance makes this analysis inconclusive. In `Maze` all methods perform slightly better than Random, but their performance is much lower than in environments that allow recovery. Additionally, none of the methods clearly outperform the others.

Discussion. Intuitively, query strategies are only useful if they detect the goal-relevant shift before it is too late for the expert to help out. This is why we use performance and not accuracy as a metric. In irrecoverable GMG, this constraint becomes much stricter than in the traditional GMG environments: detecting the shift after the proxy goal is reached is futile, because the episode has already failed. A simple time-based heuristic will fundamentally struggle with this, since waiting for the novice to fail no longer leaves room for recovery. This suggests a related failure mode across methods: the signals they rely on become informative only after proxy pursuit has already led to failure. Around the threshold values that lead to meaningful AFHP numbers in this setting, the query strategies seem to not reliably correlate with whether the novice will pursue the proxy goal.

Importantly, in both environments evidence that the distribution has shifted is in principle available before failure: in `Coinrun`, a misplaced coin and an empty final tile can be observed before the proxy goal is attained and in `Maze` the moved cheese location is visible from the start. The low performance we observe, therefore, suggests that the query strategies are not using the appropriate evidence.

Table 2: Area Under the Curve (AUC) for Average Return across Ask-For-Help Percentage (AFHP) when proxy goals cause failure.

(a) `Coinrun` AUC results.

Method	AUC (Median [IQR])
PartialOracle	**0.745 [0.721, 0.768]**
Heuristic	0.457 [0.384, 0.528]
Ensemble	**0.475 [0.450, 0.507]**
MaxProb	0.304 [0.256, 0.351]
MaxLogit	0.421 [0.369, 0.486]
ImageSVDD	0.465 [0.391, 0.531]
LatentSVDD	0.471 [0.381, 0.543]

(b) `Maze` AUC results.

Method	AUC (Median [IQR])
PartialOracle	**0.741 [0.737, 0.747]**
Heuristic	0.566 [0.560, 0.574]
Ensemble	0.567 [0.560, 0.573]
MaxProb	0.565 [0.552, 0.579]
MaxLogit	**0.583 [0.566, 0.601]**
ImageSVDD	0.539 [0.524, 0.556]
LatentSVDD	0.568 [0.558, 0.580]

6 Conclusion

We compare the feasibility of using a variety of established methods as query strategies to counter GMG in the ask-for-help setting. We find that expert deferral can substantially improve performance in standard GMG benchmarks, but that this improvement depends on recoverability: the novice can pursue the proxy goal without irreversibly preventing later success. This makes delayed strategies, including simple time-based heuristics, surprisingly effective. The mitigation also depends on an expert that can recover from novice-induced mid-episode states, not merely solve tasks from standard starts.

When we modify the benchmarks so that proxy pursuit causes immediate failure, the apparent success of existing query strategies largely disappears. This indicates a gap in current GMG evaluation: benchmarks motivated by dangerous proxy-goal pursuit often still allow recovery after the proxy is reached. Future work should expand the study of cases in which the learned proxy goal incurs negative reward or irreversibly impedes the fulfillment of the intended goal. We encourage expanding the study of mitigations to a broader range of environments, including existing ones such as those introduced by Shah et al. (2022). The methods we study are designed for traditional uncertainty estimation and score only the current state, but do not explicitly model whether the novice is likely to pursue a proxy goal, whether that pursuit will be irreversible, or whether expert intervention is still useful. Query strategies will likely have to be explicitly developed for the GMG setting.

Limitations and Future Work.

We note that, in our one-way per-episode handover protocol, successful OOD detection and timing are explicitly confounded. In practice, an OOD state may be detectable long before expert control is necessary, as in `Maze`, where the shifted goal is visible immediately but the expert is only needed once the novice's path diverges. While actively delaying the handover to the critical point is not necessary from a safety perspective, it nonetheless could greatly decrease the burden on the expert in some tasks.

In this work, we focused on the challenging scenario of zero-shot deciding whether to request expert assistance. While continual learning might come with new safety considerations, integrating a way to learn from the deployment-time expert feedback represents a natural extension. Collected expert feedback could be used to improve both the agents and the query strategy's decisions.

Similarly, we assume no a priori knowledge of which OOD situations might occur or of prior beliefs about tasks. It is conceivable that real-world GMG problems exists that are difficult to solve zero-shot, but can be mitigated sufficiently under the right priors.

Broader Impact Statement

This work tackles the problem of goal misgeneralization, a failure mode in which an AI system capably pursues an unintended proxy goal under distribution shift. We believe this phenomenon is an understudied obstacle to deploying capable agents in open-ended, real-world environments, particularly because such failures may not be detectable from training-time performance alone. By empirically studying whether asking an expert for help can mitigate goal misgeneralization, we hope to contribute to deployment-time safety mechanisms.

Our results suggest that deferring control in unfamiliar situations can improve outcomes without requiring retraining, which is particularly relevant for large, pre-trained models whose training pipelines cannot be easily modified. At the same time, this work highlights the limitations of naively using traditional OOD detection methods for this task.

Additionally, our results underscore the importance of studying goal misgeneralization when proxy goal recovery is not possible and highlight the limitations of existing evaluation suites and mitigation approaches. We hope this work supports the long-term goal of developing AI systems that remain harmless and corrigible under inevitable distribution shifts, allowing for safe deployment.

References

Abdel Sadek, Karim, Matthew Farrugia-Roberts, Usman Anwar, Hannah Erlebach, Christian Schroeder de Witt, David Krueger, and Michael D Dennis (2025). "Mitigating Goal Misgeneralization via Minimax Regret". In: *Reinforcement Learning Journal* 6, pp. 1200–1246.

Armstrong, Stuart, Alexandre Maranhão, Oliver Daniels-Koch, Patrick Leask, and Rebecca Gorman (2023). *CoinRun: Solving Goal Misgeneralisation.* arXiv: `2309.16166 [cs.AI]`.

Barj, Houda Nait El and Theophile Sautory (2024). *Reinforcement Learning from LLM Feedback to Counteract Goal Misgeneralization.* arXiv: `2401.07181 [cs.LG]`.

Black, Kevin, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky (2025). "π_0: A Vision-Language-Action Flow Model for General Robot Control". In: *Proceedings of Robotics: Science and Systems.* Los Angeles, CA, USA.

Christiano, Paul F, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei (2017). "Deep Reinforcement Learning from Human Preferences". In: *Advances in Neural Information Processing Systems.* Vol. 30. Curran Associates, Inc.

Clouse, Jeffery A. (1996). "On Integrating Apprentice Learning and Reinforcement Learning". PhD thesis. University of Massachusetts Amherst.

Cobbe, Karl, Chris Hesse, Jacob Hilton, and John Schulman (2020). "Leveraging Procedural Generation to Benchmark Reinforcement Learning". In: *Proceedings of the 37th International Conference on Machine Learning.* Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 2048–2056.

Danesh, Mohamad H., Khanh Xuan Nguyen, Tu Trinh, and Benjamin Plaut (2025). "YRC-Bench: A Benchmark for Learning to Coordinate with Experts". In: *Transactions on Machine Learning Research.*

Garcia, Javier and Fernando Fernandez (2012). "Safe Exploration of State and Action Spaces in Reinforcement Learning". In: *Journal of Artificial Intelligence Research* 45, pp. 515–564.

– (2015). "A Comprehensive Survey on Safe Reinforcement Learning". In: *Journal of Machine Learning Research* 16.42, pp. 1437–1480.

Geirhos, Robert, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann (2020). "Shortcut learning in deep neural networks". In: *Nature Machine Intelligence* 2.11, pp. 665–673.

Guo, Daya et al. (2025). "DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning". In: *Nature* 645.8081, pp. 633–638.

Han, Songqiao, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao (2022). "ADBench: Anomaly Detection Benchmark". In: *Advances in Neural Information Processing Systems Datasets and Benchmarks Track.*

Hendrycks, Dan, Steven Basart, Mantas Mazeika, Andy Zou, Joseph Kwon, Mohammadreza Mostajabi, Jacob Steinhardt, and Dawn Song (2022). "Scaling Out-of-Distribution Detection for Real-World Settings". In: *Proceedings of the 39th International Conference on Machine Learning.* Ed. by Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, pp. 8759–8773.

Hendrycks, Dan and Kevin Gimpel (2017). "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks". In: *International Conference on Learning Representations.*

Hoque, Ryan, Ashwin Balakrishna, Ellen Novoseller, Albert Wilcox, Daniel S. Brown, and Ken Goldberg (2022). "ThriftyDAgger: Budget-Aware Novelty and Risk Gating for Interactive Imitation Learning". In: *Proceedings of the 5th Conference on Robot Learning.* Vol. 164. Proceedings of Machine Learning Research. PMLR, pp. 598–608.

Ji, Jiaming, Tianyi Qiu, Boyuan Chen, Jiayi Zhou, Borong Zhang, Donghai Hong, Hantao Lou, Kaile Wang, Yawen Duan, Zhonghao He, Lukas Vierling, Zhaowei Zhang, Fanzhi Zeng, Juntao Dai, Xuehai Pan, Hua Xu, Aidan O'Gara, Kwan Ng, Brian Tse, Jie Fu, Stephen Mcaleer, Yanfeng Wang, Mingchuan Yang, Yunhuai Liu, Yizhou Wang, Song-Chun Zhu, Yike Guo, Yaodong Yang, and Wen Gao (2025). "AI Alignment: A Contemporary Survey". In: *ACM Computing Surveys* 58.5.

Kaufmann, Timo, Paul Weng, Viktor Bengs, and Eyke Hüllermeier (2025). "A Survey of Reinforcement Learning from Human Feedback". In: *Transactions on Machine Learning Research*.

Kelly, Michael, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer (2019). "HG-DAgger: Interactive Imitation Learning with Human Experts". In: *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8077–8083.

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., pp. 6402–6413.

Langosco, Lauro Langosco Di, Jack Koch, Lee D Sharkey, Jacob Pfau, and David Krueger (2022). "Goal Misgeneralization in Deep Reinforcement Learning". In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162. Proceedings of Machine Learning Research. PMLR, pp. 12004–12019.

Mosqueira-Rey, Eduardo, Elena Hernández-Pereira, David Alonso-Ríos, José Bobes-Bascarán, and Ángel Fernández-Leal (2023). "Human-in-the-loop machine learning: a state of the art". In: *Artificial Intelligence Review* 56.4, pp. 3005–3054.

Nguyen, Khanh and Hal Daumé III (2019). "Help, Anna! Visual Navigation with Natural Multimodal Assistance via Retrospective Curiosity-Encouraging Imitation Learning". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Ouyang, Long, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe (2022). "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems*. Vol. 35. Curran Associates, Inc., pp. 27730–27744.

Pathak, Abhay Kumar, Manjari Gupta, and Garima Jain (2026). "Unmasking the Clever Hans effect in AI models: shortcut learning, spurious correlations, and the path toward robust intelligence". In: *Frontiers in Artificial Intelligence* 8, p. 1692454.

Quiñonero-Candela, Joaquin, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence (2008). *Dataset Shift in Machine Learning*. The MIT Press.

Reddy, Siddharth, Anca Dragan, and Sergey Levine (2018). "Shared Autonomy via Deep Reinforcement Learning". In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania.

Retzlaff, Carl Orge, Srijita Das, Christabel Wayllace, Payam Mousavi, Mohammad Afshari, Tianpei Yang, Anna Saranti, Alessa Angerschmid, Matthew E. Taylor, and Andreas Holzinger (2024). "Human-in-the-Loop Reinforcement Learning: A Survey and Position on Requirements, Challenges, and Opportunities". In: *Journal of Artificial Intelligence Research* 79.

Ruff, Lukas, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft (2018). "Deep One-Class Classification". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 4393–4402.

Scerri, P., D. V. Pynadath, and M. Tambe (2002). "Towards Adjustable Autonomy for the Real World". In: *Journal of Artificial Intelligence Research* 17, pp. 171–228.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). *Proximal Policy Optimization Algorithms*. arXiv: `1707.06347 [cs.LG]`.

Shah, Rohin, Vikrant Varma, Ramana Kumar, Mary Phuong, Victoria Krakovna, Jonathan Uesato, and Zac Kenton (2022). *Goal Misgeneralization: Why Correct Specifications Aren't Enough For Correct Goals*. arXiv: `2210.01790 [cs.LG]`.

Stiennon, Nisan, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano (2020). "Learning to summarize with human feedback". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 3008–3021.

Tesla, Inc. (2025). *Full Self-Driving (Supervised)*. Official support documentation; FSD is an SAE Level 2 system requiring active driver supervision. URL: `https://www.tesla.com/support/fsd` (visited on 06/05/2026).

Wang, Jie, Matthew Leonard, Kostas Daniilidis, Dinesh Jayaraman, and Edward S. Hu (2025). *Evaluating pi0 in the Wild: Strengths, Problems, and the Future of Generalist Robot Policies*. URL: `https://penn-pal-lab.github.io/pi0-Experiment-in-the-Wild`.

Waymo LLC (2024). *Fleet Response: Lending a Helpful Hand to Waymo's Autonomously Driven Vehicles*. Waymo official blog. URL: https://waymo.com/blog/2024/05/fleet-response/ (visited on 06/05/2026).

Wu, Xingjiao, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He (2022). "A survey of human-in-the-loop for machine learning". In: *Future Generation Computer Systems* 135, pp. 364–381.

Ye, Wenqian, Luyang Jiang, Eric Xie, Guangtao Zheng, Yunsheng Ma, Xu Cao, Dongliang Guo, Daiqing Qi, Zeyu He, Yijun Tian, Christopher W. Porter, Megan Coffee, Zhe Zeng, Sheng Li, Ziran Wang, Ting-Hao Kenneth Huang, James Matthew Rehg, Henry Kautz, and Aidong Zhang (2026). "The Clever Hans Mirage: A Comprehensive Survey on Spurious Correlations in Machine Learning". In: *Transactions on Machine Learning Research*.

Zhao, Yue, Zain Nasrullah, and Zheng Li (2019). "PyOD: A Python Toolbox for Scalable Outlier Detection". In: *Journal of Machine Learning Research* 20.96, pp. 1–7.

Ziegler, Daniel M., Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving (2019). *Fine-Tuning Language Models from Human Preferences*. arXiv: 1909.08593 [cs.CL].

A Setup and Experiment Details

A.1 RL Environments

`Coinrun` and `Maze`, are illustrated in Figure 2. In `Coinrun`, the goal is to traverse a level in the side-view, evade enemies, and collect a coin. The levels' layout of floors and ledges is always procedurally generated, but the goal location, a coin, is always right at the end of the level in ID levels. In OOD levels, the coin is instead randomly placed in the level (but not at the far left or far right).

In the `Maze` environment, the agent needs to traverse a maze to reach a piece of cheese. The size and layout of the maze are procedurally generated, with the cheese always placed in the top-right corner in the ID version and randomly in the OOD version.

In both environments, completing the level by reaching the goal object awards a reward of 10. If the agent fails to reach the goal object due to a timeout (1000 timesteps for `Coinrun`, 500 timesteps for `Maze`), or by dying (only in `Coinrun`), it receives a reward of 0.

A.2 Hyperparameters For Novice and Expert RL Training

Table 3: Hyperparameters in the traditional setup and the robust EXPERT setup used in `Maze`.

Hyperparameter	**Traditional Setup**	**Robust Setup (`Maze`)**
ENVIRONMENT MODE	HARD	HARD
γ	.999	.999
λ	.95	.95
LEARNING RATE	0.0005	0.0005
# TIMESTEPS PER ROLLOUT	256	256
EPOCHS PER ROLLOUT	3	3
# MINIBATCHES PER EPOCH	8	8
MINIBATCH SIZE	2048	2048
ENTROPY BONUS (k_H)	.01	.01
PPO CLIP RANGE	.2	.2
LEARNING RATE	5×10^{-4}	5×10^{-4}
# PARALLEL ENVIRONMENTS	256	256
TOTAL TIMESTEPS	200M	**400M**
ARCHITECTURE	Impala	Impala

A.3 RL Policy Training Results

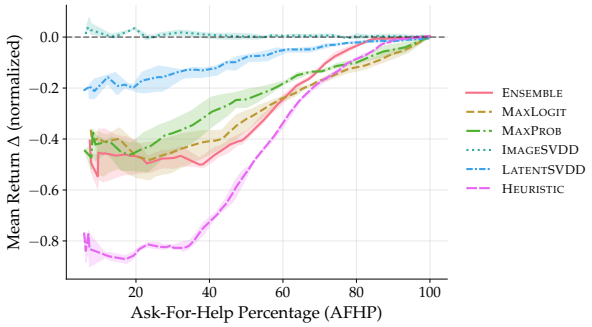
Table 4: Mean returns of NOVICE and EXPERT RL policies on ID and OOD levels, as well as the mean across all eval levels. The bolded returns indicate the extrema of the range which is used to normalize the AUC metrics. Note that the naive setting only changes the EXPERT, which is why the NOVICE is identical to `Maze` (robust).

Environment	Level Type	NOVICE	EXPERT
(A) `Maze` (robust)	ID	9.92	9.95
	OOD	3.37	9.89
	Mean	**6.64**	**9.92**
(B) `Coinrun`	ID	9.24	9.28
	OOD	7.25	9.74
	Mean	**8.24**	**9.51**
(C) `Maze` (naive)	ID	9.92	9.61
	OOD	3.37	9.24
	Mean	**6.64**	**9.42**

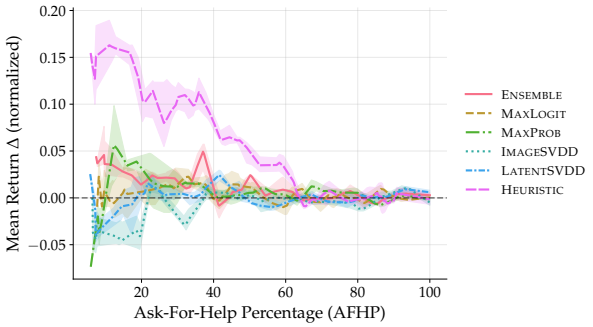
A.4 DeepSVDD training setup.

To train the DeepSVDD OOD detectors, we collect observations in the environment by rolling out the primary agent after training. We collect observations for 1024 unique levels, creating datasets of, on average, more than 100,000 observations in `Coinrun` and more than 40,000 in `Maze` across the four parallel experiments. We train with the default hyperparameters in the PyOD (Zhao et al. 2019) library and compare two variants of DeepSVDD training: one that trains on a dataset of environment observations (IMAGESVDD) and one that trains on latent representations encoded by the primary policy (LATENTSVDD).

B Maze Evaluation With Traditional Setup



(a) Performance difference in `Maze` when switching mid-level.



(b) Performance difference in `Maze` when resetting the timestep budget at the handover time.

As discussed in Section 5.2, an expert trained with the traditional setup in `Maze` yields a brittle policy whose performance deteriorates when run from non-starting states. In this section, we provide additional experiments analyzing the `Maze` expert trained with the traditional setup that `Coinrun` uses. Intuitively, despite the expert being able to solve the maze consistently when evaluated in isolation, deferring to the expert might fail for reasons such as the expert somehow being in a worse position at the time of the handover. To understand this better, we collect those levels where a query strategy asked for help and compare the performance achieved after the handover with the performance achieved by the expert when rolled out from the starting state of the same levels. We find that this decreases expert performance for some query strategies (see Figure 7a). This effect is particularly pronounced for the non-SVDD-based methods.

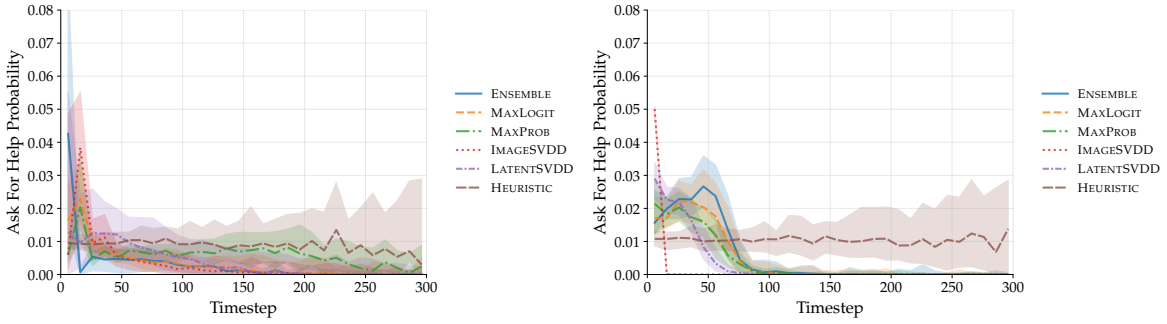
One explanation could be that when the query strategy asks for help, some time has already passed, leaving insufficient time to successfully solve the maze. To study this, we perform an additional study where the timestep budget is reset at the time of the handover, giving the expert the same amount of time as it would have if it were rolled out from the start of the level. Figure 7b shows the difference in returns, where positive values mean that increasing the timestep budget positively influenced the return. For exactly those levels where the novice asked for help, we record the mean return achieved with and without step budget resets. The returns are normalized by the expert and weak performance on each specific subset of levels, such that the relative return difference shows how much the reduction in timesteps explains the difference in return. We find that while at lower ask-for-help percentages (AFHP) the reduced budget does influence the final return, the effect does not explain the large decrease we see, especially for higher AFHP. This corroborates our theory that the main problem with our initial maze expert is the general brittleness in diverse starting states.

B.1 Changes to Maze Training

To mitigate the findings in Section B, for all experiments outside Section 5.2, we change the training setup to get a more robust expert. We use identical hyperparameters to the naive expert, and the expert in `Coinrun`, except for two modifications: We randomize the starting locations in `Maze` during training, whereas the traditional version has every agent start in the lower-left corner. Additionally, we train the expert for twice as many timesteps to give it more time to learn in this more challenging training setting. This aims to ensure the expert is able to solve the maze from any starting position.

C Quantitative Comparison of Query Strategy Deferral Probability

How do the other query strategies behave compared to Heuristic in the original, recoverable environments? Do the better ones use a similar strategy—slightly inferior, as evidenced by their lower AUC? Or do they behave fundamentally differently? To better understand this, we determine the per-timestep probability of



(a) `Coinrun` (b) `Maze`

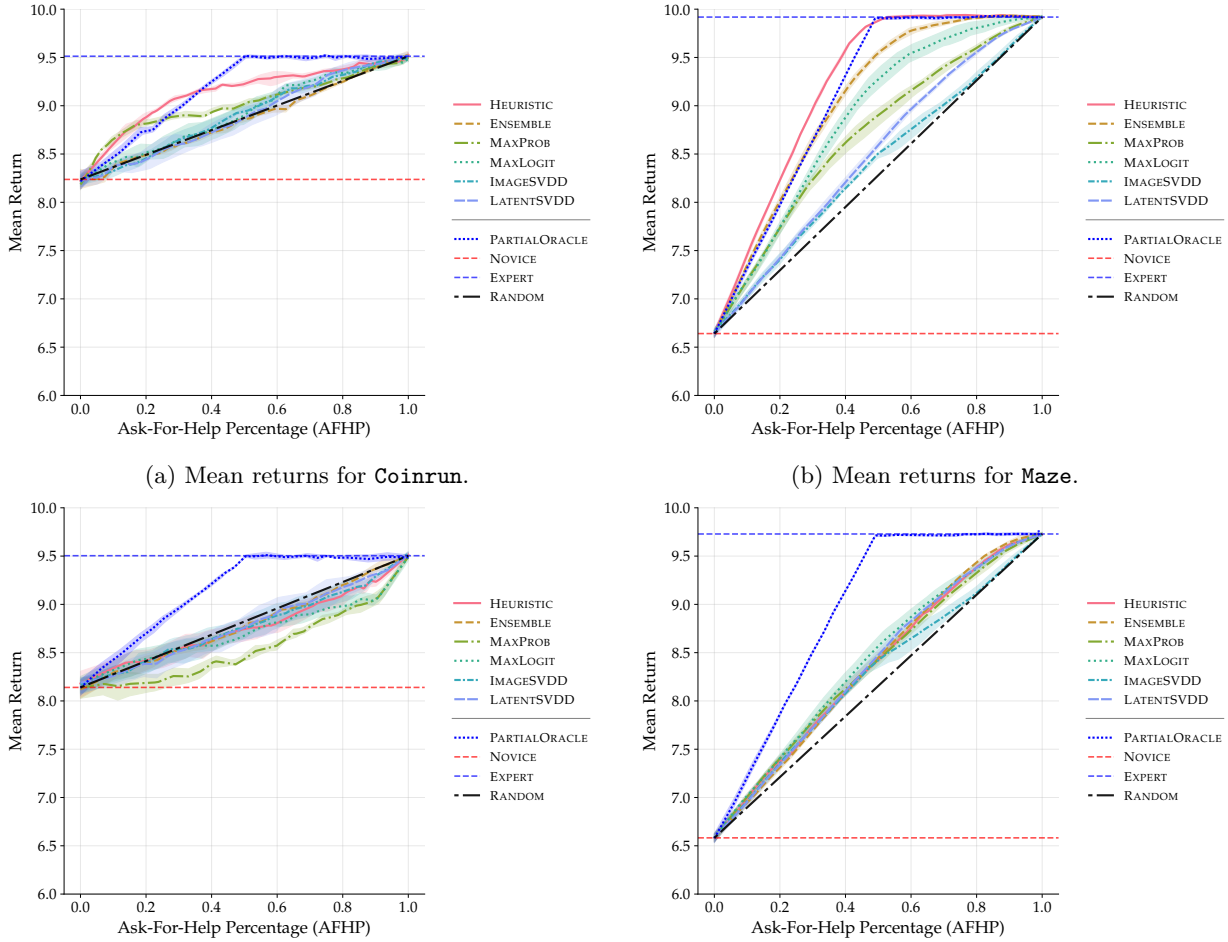
Figure 8: The average probability that a method will ask for help at a certain timestep, conditioned on the agent making it to that timestep. We calculated a binned mean and average over three independent experiments. We show the standard error across all experiments for each bin, shaded in the figure. We find that methods that tend to ask more at later timesteps tend to have better ask-for-help performance. For readability, we crop the plot at timestep 300 and ask-for-help-probability 0.08. Note that for higher timesteps, fewer runs make it that far, which causes higher variances to the right of the plot.

a given query strategy asking for help, conditioned on reaching that timestep. Figure 8 shows this ask-for-help-probability for both `Coinrun` and `Maze`. Since the exact curves differ depending on the AFHP, we plot them for AFHP = 50 as an example.[5] As expected, Heuristic is close to a straight line, as it asks for help with constant probability at every timestep. The distribution's variance increases towards the end, as fewer samples reach later timesteps without ending the level and asking for help. We observe that all methods except Heuristic have their probability mass skewed toward earlier timesteps and tend to zero as timesteps progress.

In `Coinrun`, MaxProb, which is the second-best query strategy after Heuristic, stays above zero probability mass for longer before going down to zero. Similarly, in `Maze`, the top two methods after Heuristic, Ensemble and MaxLogit, show a similar behavior, keeping more probability mass above zero for higher timestep counts. It is possible that these methods achieve superior results because they take longer to ask for help than others, which biases them toward asking for help in longer episodes, which are more likely to be OOD, just like Heuristic. When pursuing the proxy goal is not catastrophic, waiting to see how many timesteps pass without the true goal being achieved is a superior strategy to guessing earlier.

[5]Note that the AFHP is a derived value, continuous value, so it is impossible to select runs that achieve exactly 50%. We select thresholds that lead to AFHP closest to 50, and for different methods AFHP values will not be identical. Consequently, absolute ask-for-help probabilities should be interpreted qualitatively and cannot be compared exactly.

D Unnormalized Results Figures



(a) Mean returns for `Coinrun`.

(b) Mean returns for `Maze`.

(c) Mean returns for `Coinrun` when pursuit of the proxy goal leads to failure.

(d) Mean returns for `Maze` when pursuit of the proxy goal leads to failure.

Figure 9: **Unnormalized mean returns.** We compare the return of six query strategies in `Coinrun` (left) and `Maze` (right) at test time. The top row shows results in the traditional environment setup. The bottom row in the modified version where pursuit of the proxy goal leads to failure.