Beyond Fixed Tasks: Unsupervised Environment Design for Task-Level Pairs

Daniel Furelos-Blanco[*]
Imperial College London

Charles Pert
Imperial College London

Frederik Kelbel
Imperial College London

Alex F. Spies
Imperial College London

Alessandra Russo
Imperial College London

Michael Dennis[†]
Google DeepMind

Abstract

Training general agents to follow complex instructions (*tasks*) in intricate environments (*levels*) remains a core challenge in reinforcement learning. Random sampling of task-level pairs often produces unsolvable combinations, highlighting the need to co-design tasks and levels. While unsupervised environment design (UED) has proven effective at automatically designing level curricula, prior work has only considered a fixed task. We present ATLAS (Aligning Tasks and Levels for Autocurricula of Specifications), a novel method that generates *joint autocurricula* over tasks and levels. Our approach builds upon UED to automatically produce solvable yet challenging task-level pairs for policy training. To evaluate ATLAS and drive progress in the field, we introduce a *benchmark* suite that models tasks as reward machines in Minigrid levels. Experiments demonstrate that ATLAS vastly outperforms random sampling approaches, particularly when sampling solvable pairs is unlikely. We further show that mutations leveraging the structure of both tasks and levels accelerate convergence to performant policies.

1 Introduction

Training generally-capable agents that follow diverse instructions (*tasks*) in varied environments (*levels*) is a central challenge in reinforcement learning [39]. This dual complexity emerges across domains—from cooking agents executing recipes in different kitchens, to navigation agents following directives through unfamiliar cities. To generate this open-ended complexity, methods like EUREKA [34], OMNI-EPIC [17], and others [28, 56] have often relied on LLM-driven code-generation.

However, there has been growing interest in an alternative approach of expressing tasks through *formal languages*, including temporal logics [22, 29, 43, 52] and finite-state machines [54, 55]. Unlike natural language [33], formal languages offer unambiguous semantics and precise progress tracking, making them well-suited for generalization across task-level pairs. These approaches typically train agents by sampling from uninformed task-level distributions, a strategy known as *domain randomization* [DR; 45, 49].

DR can succeed when sampled task-level pairs are mostly solvable. However, as the number of task-level combinations grows, the proportion of solvable pairs tends to decrease. Even among solvable pairs, DR generates samples of *arbitrary difficulty*, causing unsolvable or overly challenging levels to dominate training. This raises the question: *How can we train agents effectively from solvable yet appropriately challenging task-level pairs?*

[*]Correspondence to `d.furelos-blanco18@imperial.ac.uk`.
[†]Contributed in an advisory capacity.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Scaling Environments for Agents (SEA).

Unsupervised environment design [UED; 16] has demonstrated success in automatically generating *level* curricula, producing solvable and progressively more challenging levels that enable effective generalization. However, existing UED methods only generate curricula over levels for a *fixed task*, yet general agents require both.

We introduce ATLAS (Aligning Tasks and Levels for Autocurricula of Specifications), which extends UED's principled curriculum generation to both tasks and levels, ensuring that agents train on solvable yet appropriately challenging pairs. We express tasks as *reward machines* [RMs; 50]—finite-state machines that compactly represent reward functions. Our *contributions* include:

1. **Joint task-level autocurricula.** ATLAS co-designs tasks and levels to generate aligned curricula, ensuring solvable yet challenging pairs. A policy network, conditioned on graph embeddings of the RMs, learns effectively from the resulting curriculum.
2. **Structure-aware task mutations.** We leverage RM structure to guide task mutations while co-evolving levels, achieving faster convergence than approaches that solely curate solvable but challenging random samples.
3. **Benchmark.** We introduce a benchmark combining RM tasks with Minigrid levels [11] that, unlike previous work, enables evaluating generalization from rarely solvable task-level pairs.

Our *experiments* show that ATLAS generates joint task-level curricula that enable agents to master increasingly complex levels (with more rooms and objects) and tasks (with more RM states). By prioritizing solvable pairs at the frontier of agent capability, ATLAS consistently outperforms DR across diverse benchmarks. Interestingly, forming curricula by repeatedly mutating tasks and levels from a simple starting point can result in policies with comparable performance to those based on random sampling.

2 Background

2.1 Unsupervised Environment Design

Unsupervised environment design [UED; 16] aims to generate training environments that adapt to an agent's capabilities, inducing an *autocurriculum*. The key is to *parameterize* RL environments (e.g., grid size) and adapt these parameters during training. The goal is to train agents that generalize across all possible parameterizations.

Formally, UED problems are *underspecified partially observable Markov decision processes* [UPOMDPs; 16]. A standard POMDP is a tuple $\mathcal{M} = \langle A, O, S, \mathcal{T}, \mathcal{I}, \mathcal{R}, \gamma \rangle$ where A is a set of actions, O is a set of observations, S is a set of states, $\mathcal{T} : S \times A \to \Delta(S)$ is the transition function, $\mathcal{I} : S \to O$ is the observation function, $\mathcal{R} : S \to \mathbb{R}$ is the reward function, and γ is the discount factor. UPOMDPs augment POMDPs with a *parameter space* Θ, which represents the space of all environment configurations. Each parameterization $\theta \in \Theta$ instantiates a fully specified POMDP $\mathcal{M}^\theta$, often called *level* [14, 24]. The expected discounted return (or *value*) of a *policy* π in level $\mathcal{M}^\theta$ is $V^\theta(\pi) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t]$, where T is a horizon. The goal is to produce a sequence of distributions over Θ that maximizes the policy's value across levels.

2.2 Methods for UED

UED methods often frame curriculum design as a game between two players: a *teacher*, which proposes levels, and a *student*, which trains on them. The teacher generates levels by maximizing a *utility score*. We review two scoring strategies: *constant* and *regret-based* scoring.

Constant. All levels have equal utility, ignoring the student's performance. Parameters θ are uniformly sampled from the parameter space Θ. This strategy is commonly known as *domain randomization* [DR; 45, 49].

Regret-Based. The regret for a level $\mathcal{M}^\theta$ is the value gap $V^\theta(\pi^*) - V^\theta(\pi)$ between the optimal policy π^* and the current policy π. Unlike DR, regret scoring deprioritizes unsolvable levels and focuses on those at the frontier of the agent's capabilities. Theoretically, a teacher that maximizes regret induces a minimax regret student policy at equilibrium [16].

Since finding the optimal policy is intractable, UED methods resort to regret approximations. For example, MaxMC [23] uses the highest undiscounted return seen so far, $R^{\theta}_{\max}$, as a proxy for optimal performance in $\mathcal{M}^{\theta}$ and computes regret as $(1/T)\sum_{t=0}^{T} R^{\theta}_{\max} - V^{\theta}(o_t)$, where $V^{\theta}(o_t)$ is the value of the observation at time t.

Prioritized Level Replay [PLR; 24] curates a *buffer* of high-regret levels. At each episode, PLR chooses between (i) with probability p, replaying levels from the buffer and updating their estimates, and (ii) with probability $1-p$, sampling new levels via DR and adding them to the buffer. In both cases, the student policy trains on the selected levels.

In this paper, we consider two extensions of PLR. *Robust PLR* [PLR$^{\perp}$; 23] trains the student only on replayed levels. *ACCEL* [40] extends PLR$^{\perp}$ by *mutating* the last replayed levels (e.g., moving an object) before adding them to the buffer, thereby exploring the neighborhood of high-regret levels rather than solely relying on random sampling.

2.3 Reward Machines

Reward machines [RMs; 50] are finite-state machines for specifying reward functions using high-level propositional events. They provide a flexible task representation, supporting derivations from other formal languages [9] and indirect mappings from natural language [32, 51].

Formally, an RM is a tuple $\langle U, P, \delta, \mathcal{R}, u_0, u_A \rangle$, where U is a finite set of states; P is a finite set of propositions that forms the *alphabet* of the RM; $\delta : U \times 2^P \to U$ is the state-transition function, which maps an RM state and a subset of propositions into an RM state; $\mathcal{R} : U \times U \to \mathbb{R}$ is the reward-transition function, which maps an RM state pair into a reward; $u_0 \in U$ is the initial state of the RM; and $u_A \in U$ is the accepting state of the RM, which represents the task's completion.

We refer to propositions sets $L \in 2^P$ as *labels*. These are produced by a *labeling function* $\mathcal{L} : O \to 2^P$ from environment observations. Given an RM state u and a label L, the RM transitions to $u' = \delta(u, L)$ and emits reward $\mathcal{R}(u, u')$.

By enabling RMs to call each other, RMs can be composed into *hierarchies* [HRMs; 19]. This modular structure supports task decomposition and reusability across HRMs. For such HRMs, a single RM is designated as the *root*, initiating execution and terminating upon reaching its accepting state; other RMs return control to their caller upon completion.

3 A Task-Level Generalization Benchmark

We introduce a benchmark for problem-conditioned RL, where a *problem specification* (hereafter, *problem*) is a tuple consisting of a *task*, an instruction the agent must follow, and a *level*, the environment instance in which the agent acts.

The core *challenge* lies in generalizing from task-level pairs that are *rarely solvable*. This stems from the large combinatorial space of possible problems, where tasks may require interactions that are infeasible in a level (e.g., picking up a red ball in a level containing no balls). Prior benchmarks for problem-conditioned RL ensure randomly sampled pairs are highly solvable (see Section 6).

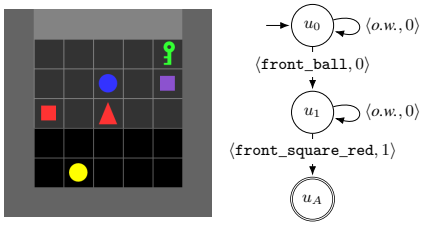


Figure 1: A *problem* consisting of a Minigrid *level* and an RM *task* for *"go to a ball, then go to a red square"*.

Our benchmark targets two key fronts. First, it probes the limits of *problem-conditioned methods* that train solely from DR-sampled problems. Second, it broadens the scope of *UED*, enabling settings where tasks are not fixed but must be co-designed with levels. To support these goals, the benchmark provides 150 *hand-designed problems* and modular samplers for tasks, levels, and joint generation (see Appendix A).

3.1 Levels

Our benchmark instantiates *levels* as Minigrid environments [11]. A level consists of a grid containing *objects* (keys, squares, balls, doors) with different *colors* (red, green, blue, purple, yellow, gray).

Doors have three *states*: locked, open, or closed, with locked doors requiring keys of matching color to open. The agent observes a 5×5 region in front of it. Figure 1 (left) shows a Minigrid level with the agent (red triangle) and its observation (highlighted area). We develop *samplers* that generate levels with a random number of rooms and objects. All objects are randomly determined. Agent and non-door objects are randomly placed. Rooms are connected via doors.

3.2 Tasks

Tasks are instantiated as HRMs that encode BabyAI instructions [10]. These instructions consist of *go to*, *open*, *pick up* and *put next* commands, each applied to specific objects, optionally conditioned on color and state. The HRM *alphabet* is defined by mapping these commands to propositions:

- $\texttt{front\_}\langle \texttt{o}_1 \rangle\texttt{\_}\langle \texttt{c}_1 \rangle\texttt{\_}\langle \texttt{s}_1 \rangle$ indicates the agent is in front of object $\texttt{o}_1$ with color $\texttt{c}_1$ and state $\texttt{s}_1$,
- $\texttt{carrying\_}\langle \texttt{o}_1 \rangle\texttt{\_}\langle \texttt{c}_1 \rangle\texttt{\_}\langle \texttt{s}_1 \rangle$ indicates the agent carries object $\texttt{o}_1$ with color $\texttt{c}_1$ and state $\texttt{s}_1$, and
- $\texttt{next\_}\langle \texttt{o}_1 \rangle\texttt{\_}\langle \texttt{c}_1 \rangle\texttt{\_}\langle \texttt{s}_1 \rangle\texttt{\_}\langle \texttt{o}_2 \rangle\texttt{\_}\langle \texttt{c}_2 \rangle\texttt{\_}\langle \texttt{s}_2 \rangle$ indicates that object $\texttt{o}_1$ with color $\texttt{c}_1$ and state $\texttt{s}_1$ is next to object $\texttt{o}_2$ with color $\texttt{c}_2$ and state $\texttt{s}_2$ within the agent's visual field,

where $\texttt{o}_i \in \{\texttt{ball}, \texttt{square}, \texttt{key}, \texttt{door}\}$, $\texttt{c}_i \in \{\texttt{red}, \texttt{green}, \texttt{blue}, \texttt{purple}, \texttt{yellow}, \texttt{gray}, \epsilon\}$, and $\texttt{s}_i \in \{\texttt{open}, \texttt{closed}, \texttt{locked}, \epsilon\}$ for $i \in \{1, 2\}$. The state $\texttt{s}_i$ is unspecified (ϵ) if $\texttt{o}_i \neq \texttt{door}$. The propositions capture each instruction type: `front` for *go to* and *open*, `carrying` for *pick up*, and `next` for *put next*. Crucially, HRMs enable sequencing and alternating formulas over these propositions, mirroring the connectors *then* and *and* in BabyAI instructions.

Figure 1 (right) shows an RM for the instruction *"go to a ball, then go to a red square"*. A reward of 1 is given upon completing the task, and 0 otherwise. Given the level on its left, the observation of the agent is mapped into the label $\{\texttt{front\_ball}, \texttt{front\_ball\_blue}, \texttt{next\_square\_purple\_key\_green}, \texttt{next\_square\_key\_green}, \texttt{next\_square\_key}, \texttt{next\_square\_purple\_key}\}$, which satisfies the transition from u_0 to u_1.

We introduce two HRM *sampling strategies*. First, the *sequential* sampler generates non-hierarchical HRMs with a single path from u_0 to u_A, where the path length is sampled from a predefined range (e.g., Figure 1). Second, the *random walk-based* sampler generates potentially hierarchical HRMs with one or more (possibly cyclic) paths from u_0 to u_A, subsuming the sequential case. Transitions are determined by a random walk over a uniformly initialized Markov transition matrix (see Appendix A.1 for details). In both strategies, each transition is labeled with a proposition p sampled uniformly from the alphabet, while its negation $\neg p$ is added to all other outgoing transitions from the same state to enforce *determinism*.

Sampled HRMs support a variety of *reward functions*, including sparse rewards (1 only when reaching u_A), step-wise rewards [1 on each transition to a new state, e.g. 51], and shaped rewards based on the distance to u_A [e.g., 8, 18].

3.3 Problem Sampling

There are two main strategies to sample a problem: *independent*, in which tasks and levels are sampled separately; and *level-conditioned*, in which task generation is constrained by the objects in the sampled level (i.e., only propositions involving those objects may label the edges in the HRM).

Level-conditioning increases the likelihood of sampling *solvable* problems, but does not guarantee it. For example, a task may require unlocking a door with an existing but unreachable key. Guaranteeing solvability is challenging, as it may require solving the problem during generation.

4 Problem-Conditioning via Autocurricula

We introduce ATLAS (Aligning Tasks and Levels for Autocurricula of Specifications), a method for learning policies that *generalize* across problem specifications by leveraging *autocurricula*—automatically adapting the training problems to match the agent's capabilities. This is critical in settings where *solvable* problems are rarely sampled.

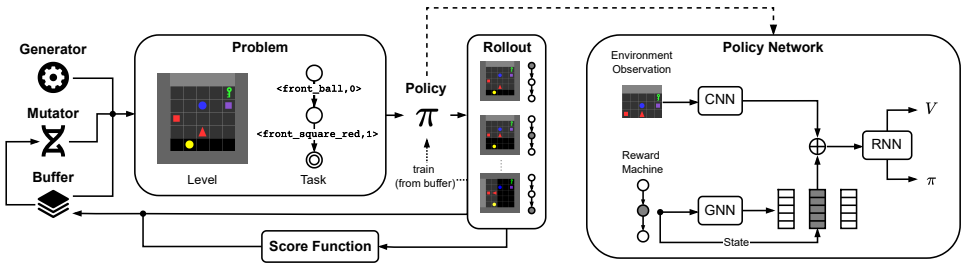


Figure 2: Overview of ATLAS instantiated with PLR$^{\perp}$ and ACCEL. The UED loop **(left)** samples problems—i.e., task-level pairs—from either a generator or a buffer of high-regret problems. ACCEL provides problems that result from mutating selected buffer problems. The policy network **(right)** processes observations via a convolutional neural network (CNN) and RM tasks via a graph neural network (GNN). The GNN produces representations for all RM states. The current state's embedding is concatenated with the CNN features and passed through a recurrent neural network (RNN) to capture history. The resulting representation is used to generate actions and value estimates. Policy rollouts are used to train the network (for buffer-sourced problems), and to compute regret scores, which determine if new problems enter the buffer or update existing ones. Unlike PLR$^{\perp}$ and ACCEL, DR trains policies only from problems produced by the generator.

Our approach extends UED beyond level generation to *jointly* adapt over both tasks and levels. By co-designing these, regret-based UED methods generate problems that are not only solvable (i.e. the task is feasible within the corresponding level) but also challenging.

Figure 2 illustrates ATLAS, which comprises two components: a problem-conditioned policy network and a UED-driven curriculum generation loop. In this work, we focus on flat HRMs (i.e., standard RMs), deferring hierarchies to future extensions.

4.1 Policy Architecture

To generalize across task-level combinations, we use an actor-critic architecture conditioned on both environment observations and RM task states. The *key* design choice is to encode RMs using a *graph neural network* (GNN). Unlike fixed embeddings (e.g., one-hot encodings of the RM state index), GNNs naturally handle varying RM topologies and edge labelings, supporting generalization across RMs. The policy is trained using PPO [47]. See Appendix C for details.

4.2 Problem Autocurricula via UED

We extend UED approaches to support joint generalization over both tasks and levels. For brevity, we refer to ATLAS instantiations of these methods as DR, PLR$^{\perp}$, and ACCEL, corresponding to the underlying UED approach. In the latter two, generalization is driven by automatically induced curricula over both levels and RM tasks.

Unlike the original ACCEL [40], which only applied level mutations, ATLAS' instantiation incorporates *task-aware mutations* leveraging the RM structure. This enables exploring a broader neighborhood of problems by modifying both tasks and levels.

A mutation consists of a sequence of edits, where the number of edits (sampled from a predefined range) and each edit type are selected uniformly at random. We define three *types of edits* (see Appendix D for examples):

Level Edits. Edits applied to the environment. In Minigrid, this includes moving the agent, adding/removing rooms, and adding/removing/replacing/moving an object.

Task Edits. Edits applied to the RM structure. These include switching a proposition and adding/removing a state.

Hindsight Edits. Inspired by *hindsight relabeling* [4], these edits generate new subproblems from partial progress. When a rollout ends in an intermediate RM state u (neither initial nor accepting),

there are two possible edits: *preceding edits* keep the original level but set u as the accepting RM state, yielding a simpler problem, and *succeeding edits.* use the reached environment state as the new level and set u as the initial RM state, creating a continuation problem. Since these edits depend on the original problem, they can only be applied as the first step in a mutation sequence.

5 Experiments

We address the following research questions:

RQ1 Joint Curriculum Effectiveness. Do approaches that generate curricula over both levels and tasks (PLR$^{\perp}$, ACCEL) outperform DR?

RQ2 Mutation Benefits. Does incorporating mutations (ACCEL) offer advantages over curation-only approaches (PLR$^{\perp}$)?

RQ3 Joint Curriculum Emergence. Does ATLAS induce autocurricula over both levels and tasks?

RQ4 Mutation Analysis. Which mutation types become most prevalent during training?

We address these questions in our main results, supported by key ablations below and extended analysis in Appendix E. The code is available at `https://github.com/spike-imperial/atlas`.

5.1 Experimental Setup

We describe the *default* training setup used in our experiments. See Appendix E.2 for details.

Generation. Levels and tasks are sampled *independently*, creating a setting in which most training problems are not solvable. *Levels* are generated by sampling (i) a number of rooms in $\{1, 2, 4, 6\}$, (ii) a number of objects from a grid-dependent range, (iii) the objects themselves, and (iv) the agent position. *Tasks* are specified as RMs generated via the sequential sampler, with path lengths randomly chosen between 1 and 5. Although our benchmark supports various reward functions (see Section 3.2), we use sparse rewards—1 on transitions to u_A and 0 otherwise—as they are general yet challenging, making them well-suited for assessing performance and curricula emergence.

Algorithms. We evaluate ATLAS instantiated with DR, PLR$^{\perp}$, and ACCEL. For ACCEL, we distinguish between (i) ACCEL, where problems are sampled using the setup above; and (ii) ACCEL-0, where sampled problems consist of single-room levels with one object and RMs with one transition, and complexity emerges solely through mutations. We apply the *mutations* described in Section 4.2, with sequence lengths sampled uniformly in the range 7–10.

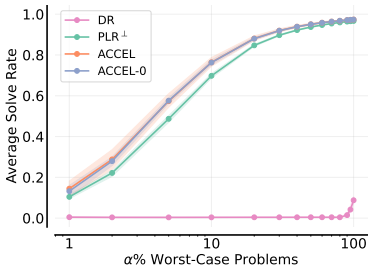
Metrics. We evaluate performance using two metrics, averaged over five seeds with 95% confidence intervals. First, we compute *conditional value at risk* [CVaR; 44], which quantifies *robustness* by reporting the solve rate for the $\alpha\%$ worst-performing problems in a large sampled set. Second, we report *test performance* using the aggregate *inter-quartile mean* [IQM; 2] of solve rates on our hand-designed evaluation set, assessing both in-distribution and out-of-distribution generalization.

5.2 Main Results

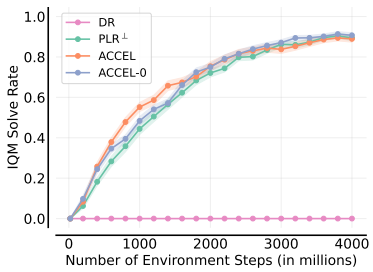
We present our core experimental results addressing the research questions outlined earlier.

Performance (RQ1, RQ2). Figure 3a displays CVaR results measuring agent *robustness*. ACCEL variants consistently outperform PLR$^{\perp}$ across most α values, particularly for worst-case scenarios (low α). At $\alpha = 100\%$ (average performance), both methods perform similarly and substantially outperform DR, which solves almost no problems.

Figure 3b shows zero-shot performance over time on our hand-designed evaluation set. While ACCEL variants and PLR$^{\perp}$ achieve comparable final results, ACCEL converges faster. Notably, ACCEL-0 achieves strong performance, demonstrating that problem complexity can emerge purely through targeted mutations, without relying on initial problem diversity. DR again underperforms due to the scarcity of solvable training problems.

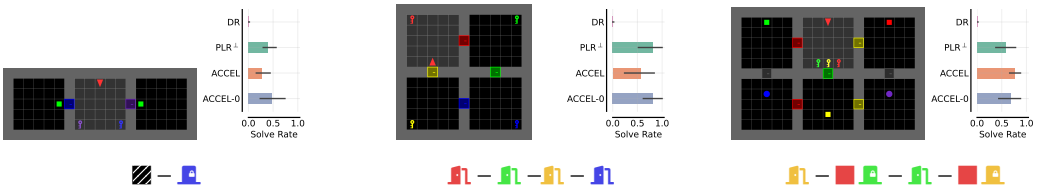


(a) CVaR of the solve rate.



(b) Aggregate zero-shot performance on the hand-designed test set.

Figure 3: Performance of ATLAS variants on (a) the worst-case problems and (b) a challenging hand-designed test set.



(a) MYOPIC (b) PATROL (c) CHOICE

Figure 4: Zero-shot performance of ATLAS on hand-designed problems. Symbol sequences represent the RM tasks. Symbols represent balls (●), squares (■), keys (🔑), and closed/locked/open/unspecified doors (■/■/■/■). Unspecified doors can match any state. Single symbols indicate `front` propositions, pairs indicate `next` propositions. Striped patterns represent an unspecified color (e.g., ▨ stands for `front_square`).

Figure 4 breaks down performance on three challenging hand-designed scenarios. MYOPIC requires long-term planning: the agent must face a square *but* it cannot be the one behind the locked blue door, as unlocking that door renders the problem unsolvable. PATROL involves navigating with underspecified instructions (i.e., carrying the keys is required to unlock the doors but not mentioned). CHOICE combines both: agents must explore the grid while ensuring some doors remain locked. In all cases, regret-based methods outperform DR by a wide margin.

The success of PLR$^{\perp}$ and ACCEL stems from filtering solvable problems. Independent sampling yields only 2.7% solvable problems per batch (see Appendix E.3), making task-level co-design essential. Both methods curate a buffer of high-regret problems, resulting in the generation of challenging yet solvable problems. Indeed, the fraction of solvable buffer problems steadily grows, eventually nearing 100% (see Appendix E.4)—interestingly, the growth is faster in ACCEL than for PLR$^{\perp}$. DR, in contrast, trains on predominantly unsolvable problems throughout.

Curriculum Analysis (RQ3). We analyze how curricula emerge across both tasks and levels. Figure 5 illustrates the evolution of buffer problems during training. Initially, PLR$^{\perp}$ and ACCEL curate buffers with simple problems—few rooms, objects, and RM states. Over time, problem complexity increases along all dimensions. ACCEL variants reach higher RM state counts than PLR$^{\perp}$, reflecting a stronger emphasis on task complexity. As expected, ACCEL-0 starts with the simplest problems (two states, one room, one object) as it only relies on mutations from these.

Figure 6 provides examples of generated training problems near the end of training. For PLR$^{\perp}$ and ACCEL, these are sampled from their respective buffers, while DR examples are drawn directly from the generator. Both PLR$^{\perp}$ and ACCEL prioritize dense six-room layouts, but ACCEL favors RMs with more states. The problems generated by PLR$^{\perp}$ and ACCEL are complex, requiring efficient exploration, implicit door unlocking, and even maintaining doors locked to preserve solvability (e.g., the gray doors in ACCEL). In contrast, DR trains on mostly unsolvable or overly complex problems, which the agent struggles to learn from.

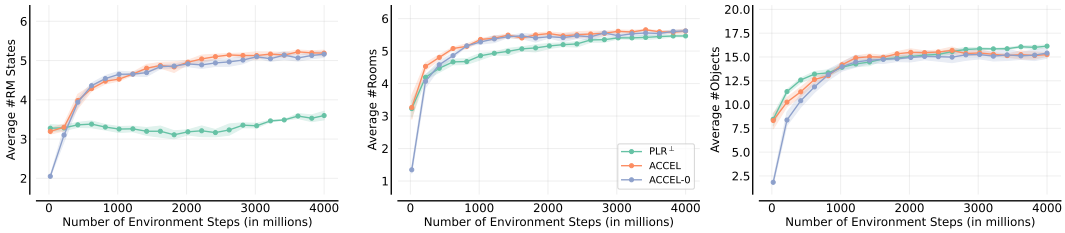


Figure 5: Emergent complexity metrics for problems in the buffer.

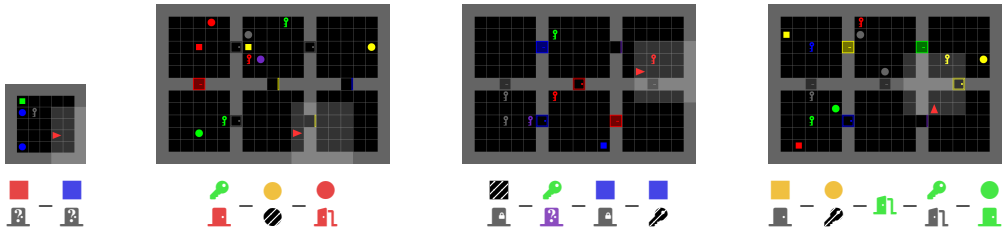


Figure 6: Generated problems by ATLAS (DR, PLR$^{\perp}$, ACCEL, ACCEL-0).

Mutation Analysis (RQ4). We observe the *average number of edits per problem* increases over training, indicating that (i) longer edit sequences are beneficial and (ii) the buffer content shifts from primarily randomly generated problems to predominantly mutated ones. This trend shows ACCEL successfully compounds complexity over time, rather than relying on random sampling alone.

We analyze how different *edit types* shape curricula by measuring their frequency in the buffers, weighted by the sampling probabilities of the problems they generate. Task and level edits contribute roughly equally. In contrast, hindsight edits are rare, likely due to limited applicability: they require rollouts to end in intermediate RM states and can only appear as the first edit. Exploring more general hindsight edits is an interesting direction for future work. See Appendix E.4 for illustrations.

5.3 Problem Sampling Ablations

In the default setup, levels and RM tasks are sampled *independently*, yielding only 2.7% solvable problems per batch. In this setting, PLR$^{\perp}$ and ACCEL vastly outperformed DR. To test the hypothesis that this gap stems from the scarcity of solvable problems, we perform an ablation with *level-conditioned* problem sampling (see Section 3.3), which increases solvability to 83.4%.

We make three key observations. First, as hypothesized, DR improves substantially, matching the CVaR and test performance (≈91.6%) of PLR$^{\perp}$ and ACCEL variants. Second, PLR$^{\perp}$ and ACCEL see only marginal gains (≈1–3%) over the independent setting, highlighting their robustness when solvable problems are rare. Third, a curriculum still emerges: for PLR$^{\perp}$, it becomes more pronounced, with RM tasks containing 4–5 states increasingly prioritized. See Appendix E.5 for further analysis.

5.4 Task Sampling Ablations

To evaluate generalization under more complex training distributions, we replace the sequential task sampler with the *random walk-based* task sampler, restricted to generate RMs as *directed acyclic graphs* (see Section 3). This class subsumes sequential tasks and introduces multiple acyclic paths from u_0 to u_A, creating richer temporal dependencies. As in the default setup, the maximum number of states is 6. We restrict comparisons to DR and PLR$^{\perp}$ under independent problem sampling.

We find that PLR$^{\perp}$ continues to substantially outperform DR for both CVaR and zero-shot performance on the hand-designed set. However, solve rates on the hand-designed evaluation set drop by ≈35% compared to the sequential setting, indicating that increased task complexity can affect overall performance. See Appendix E.6 for full results.

5.5 Mutation Ablations

We assess ACCEL's performance under two types of ablations: (i) *disabling specific edit types*, and (ii) *varying the edit sequence length*. Disabling either level or task edits significantly reduces performance, while combining both types yields the best results. This highlights the necessity of joint task-level mutations for effective training. ACCEL is sensitive to sequence length: short sequences severely hinder performance, with single-edit sequences causing a $\approx$40% drop and 3-edit sequences leading to a $\approx$15% drop. In contrast, long sequences (20 edits) have minimal effect ($\approx$2% variation). See Appendix E.7 for details.

6 Related Work

We summarize key related work here, with an extended discussion in Appendix B.

Unsupervised Environment Design (UED). To the best of our knowledge, research on UED has exclusively focused on level generation with fixed tasks. ATLAS is the first UED-based approach to co-design both tasks and levels.

Seminal UED approaches rely on regret-based utility scores [16, 23], which we also employ in ATLAS. However, recent work identifies three limitations: high-regret levels need not be diverse [30]; regret can become irreducible, causing training to stagnate [6]; and common regret approximations correlate with success rate rather than true regret [44]. To address these issues, alternative scores—diversity [30], novelty [48], and learnability, which prioritizes levels solved 50% of the time [44]—have been proposed. We hypothesize irreducible floors may emerge under any scoring function. Our benchmark supports future research using task-level variations to evaluate these limitations.

Alternative approaches to automatic level design beyond UED have been proposed. POET [53] trains populations of specialist agents using evolutionary strategies, while ATLAS trains general agents from random and co-evolved task-level pairs. PCGRL [25] frames level design as an RL problem, where levels are incrementally edited to optimize a given quality objective.

Formal Language Conditioning. Most closely related to our work, Yalcinkaya et al. [54, 55] condition policies on GNN embeddings of *automata* representing reach-avoid task sequences. While Yalcinkaya et al. [55] apply task mutations, their goal is to derive task representations, not to construct a curriculum. In contrast, ATLAS targets high-regret tasks for curriculum generation and co-evolves both tasks and levels, whereas their method only modifies tasks.

Linear temporal logic [LTL; 42] approaches typically train using formulas sampled from context-free grammars. Kuo et al. [29] and Vaezipoor et al. [52] encode task structure by embedding the formula's syntax tree through compositional RNNs and GNNs, respectively. Qiu et al. [43] and Jackermeier and Abate [22], unlike previous work and ATLAS, tackle *infinite-horizon* tasks by mapping formulas into equivalent automata whose paths determine the conditioning—the former is sequentially conditioned on each subtask along a path, while the latter is conditioned on derived reach-avoid sequences.

These approaches implement DR, sometimes with fixed curricula [22], which may collapse when solvable problems are rarely sampled (see Section 5). In contrast, ATLAS leverages regret-based UED to generate *autocurricula* of solvable yet challenging problems. Prior work with DR succeeded because their domains ensure high solvability: all propositions are observable across levels, making most tasks completable. We address this evaluation gap through a benchmark with naturally low solvability rates, revealing the limitations of DR.

7 Conclusions

We introduce ATLAS, a novel method for generating joint autocurricula over problems (task-level pairs). By co-designing tasks and levels via regret-based UED, ATLAS achieves robust generalization even when most sampled problems are unsolvable—a setting in which domain randomization fails. Additionally, ACCEL-0 achieves strong performance and builds rich curricula by repeatedly mutating initially simple problems. We contribute 150 hand-designed test problems on which we verify that these findings hold, validating our approach and providing a foundation for further research on task-level generalization.

Future Work. Our framework opens several promising research directions. First, extending the approach to other temporal formalisms—including LTL, programs, or hierarchical specifications—would demonstrate the broad applicability of joint curriculum generation via UED. Second, developing scoring functions and mutation operators that better exploit task structure could yield even more effective curricula. Finally, scaling to domains like Craftax [35] would test the effectiveness of joint curricula in richer settings.

Acknowledgments and Disclosure of Funding

We thank the reviewers and Roko Parać for their valuable feedback on this manuscript. We thank Isabella Pearce for her invaluable assistance in designing the benchmark problems, and Dugan Witherick for his essential support in setting up the computational resources provided by the Imperial College Research Computing Service (`http://doi.org/10.14469/hpc/2232`).

This work is partially supported by DEVCOM Army Research Lab under grant W911NF2220243 and EPSRC projects EP/X040518/1 and EP/Y037421/1. Charles Pert is supported by UK EPSRC grant 2760033. Frederik Kelbel is supported by UK EPSRC grant 2757464.

References

[1] R. Abboud, İ. İ. Ceylan, M. Grohe, and T. Lukasiewicz. The Surprising Power of Graph Neural Networks with Random Node Initialization. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[2] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. G. Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[3] J. Andreas, D. Klein, and S. Levine. Modular Multitask Reinforcement Learning with Policy Sketches. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[4] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight Experience Replay. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[5] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer Normalization. *arXiv preprint*, arXiv:1607.06450, 2016.

[6] M. Beukman, S. Coward, M. T. Matthews, M. Fellows, M. Jiang, M. D. Dennis, and J. N. Foerster. Refining Minimax Regret for Unsupervised Environment Design. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[7] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`.

[8] A. Camacho, O. Chen, S. Sanner, and S. A. McIlraith. Non-Markovian Rewards Expressed in LTL: Guiding Search Via Reward Shaping. In *Proceedings of the International Symposium on Combinatorial Search (SOCS)*, 2017.

[9] A. Camacho, R. Toro Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[10] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[11] M. Chevalier-Boisvert, B. Dai, M. Towers, R. Perez-Vicente, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. K. Terry. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[12] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[13] H. Chung, J. Lee, M. Kim, D. Kim, and S. Oh. Adversarial Environment Design via Regret-Guided Diffusion Models. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[14] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging Procedural Generation to Benchmark Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[15] S. Coward, M. Beukman, and J. N. Foerster. JaxUED: A simple and useable UED library in Jax. *arXiv preprint*, arXiv:2403.13091, 2024.

[16] M. Dennis, N. Jaques, E. Vinitsky, A. M. Bayen, S. Russell, A. Critch, and S. Levine. Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[17] M. Faldor, J. Zhang, A. Cully, and J. Clune. OMNI-EPIC: Open-endedness via Models of human Notions of Interestingness with Environments Programmed in Code. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[18] D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. Induction and Exploitation of Subgoal Automata for Reinforcement Learning. *Journal of Artificial Intelligence Research*, 70: 1031–1116, 2021.

[19] D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. Hierarchies of Reward Machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

[20] S. Garcin, J. Doran, S. Guo, C. G. Lucas, and S. V. Albrecht. DRED: Zero-Shot Transfer in Reinforcement Learning via Data-Regularised Environment Design. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[21] J. Godwin, T. Keck, P. Battaglia, V. Bapst, T. Kipf, Y. Li, K. Stachenfeld, P. Veličković, and A. Sanchez-Gonzalez. Jraph: A library for graph neural networks in JAX, 2020. URL `http://github.com/deepmind/jraph`.

[22] M. Jackermeier and A. Abate. DeepLTL: Learning to efficiently satisfy complex LTL specifications for multi-task RL. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[23] M. Jiang, M. Dennis, J. Parker-Holder, J. N. Foerster, E. Grefenstette, and T. Rocktäschel. Replay-Guided Adversarial Environment Design. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[24] M. Jiang, E. Grefenstette, and T. Rocktäschel. Prioritized Level Replay. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[25] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius. PCGRL: Procedural Content Generation via Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2020.

[26] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[27] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[28] M. Klissarov, M. Henaff, R. Raileanu, S. Sodhani, P. Vincent, A. Zhang, P.-L. Bacon, D. Precup, M. C. Machado, and P. D'Oro. MaestroMotif: Skill Design from Artificial Intelligence Feedback. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[29] Y. Kuo, B. Katz, and A. Barbu. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[30] W. Li, P. Varakantham, and D. Li. Generalization through Diversity: Improving Unsupervised Environment Design. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2023.

[31] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky. Graph Normalizing Flows. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[32] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah. Grounding Complex Natural Language Commands for Temporal Tasks in Unseen Environments. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2023.

[33] J. Luketina, N. Nardelli, G. Farquhar, J. Foerster, J. Andreas, E. Grefenstette, S. Whiteson, and T. Rocktäschel. A survey of reinforcement learning informed by natural language. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[34] Y. J. Ma, W. Liang, G. Wang, D. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-Level Reward Design via Coding Large Language Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[35] M. T. Matthews, M. Beukman, B. Ellis, M. Samvelyan, M. T. Jackson, S. Coward, and J. N. Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[36] J. Mu, V. Zhong, R. Raileanu, M. Jiang, N. D. Goodman, T. Rocktäschel, and E. Grefenstette. Improving Intrinsic Exploration with Language Abstractions. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[37] C. Neary, Z. Xu, B. Wu, and U. Topcu. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2021.

[38] A. Nikulin, V. Kurenkov, I. Zisman, A. Agarkov, V. Sinii, and S. Kolesnikov. XLand-MiniGrid: Scalable Meta-Reinforcement Learning Environments in JAX. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[39] OEL Team, A. Stooke, A. Mahajan, C. Barros, C. Deck, J. Bauer, J. Sygnowski, M. Trebacz, M. Jaderberg, M. Mathieu, N. McAleese, N. Bradley-Schmieg, N. Wong, N. Porcel, R. Raileanu, S. Hughes-Fitt, V. Dalibard, and W. M. Czarnecki. Open-Ended Learning Leads to Generally Capable Agents. *arXiv preprint*, arXiv:2107.12808, 2021.

[40] J. Parker-Holder, M. Jiang, M. Dennis, M. Samvelyan, J. N. Foerster, E. Grefenstette, and T. Rocktäschel. Evolving Curricula with Regret-Based Environment Design. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

[41] T. Pierrot, G. Ligner, S. E. Reed, O. Sigaud, N. Perrin, A. Laterre, D. Kas, K. Beguir, and N. de Freitas. Learning Compositional Neural Programs with Recursive Tree Search and Planning. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[42] A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 1977.

[43] W. Qiu, W. Mao, and H. Zhu. Instructing Goal-Conditioned Reinforcement Learning Agents with Temporal Logic Objectives. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[44] A. Rutherford, M. Beukman, T. Willi, B. Lacerda, N. Hawes, and J. N. Foerster. No Regrets: Investigating and Improving Regret Approximations for Curriculum Discovery. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[45] F. Sadeghi and S. Levine. CAD2RL: Real Single-Image Flight Without a Single Real Image. In *Proceedings of the Robotics: Science and Systems Conference (RSS)*, 2017.

[46] M. S. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling Relational Data with Graph Convolutional Networks. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, 2018.

[47] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint*, arXiv:1707.06347, 2017.

[48] J. Teoh, W. Li, and P. Varakantham. Improving Environment Novelty Quantification for Effective Unsupervised Environment Design. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[49] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[50] R. Toro Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[51] M. Tuli, A. C. Li, P. Vaezipoor, T. Q. Klassen, S. Sanner, and S. A. McIlraith. Learning to Follow Instructions in Text-Based Games. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[52] P. Vaezipoor, A. C. Li, R. Toro Icarte, and S. A. McIlraith. LTL2Action: Generalizing LTL Instructions for Multi-Task RL. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[53] R. Wang, J. Lehman, J. Clune, and K. O. Stanley. Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. *arXiv preprint*, arXiv:1901.01753, 2019.

[54] B. Yalcinkaya, N. Lauffer, M. Vazquez-Chanlatte, and S. Seshia. Automata Conditioned Reinforcement Learning with Experience Replay . In *Proceedings of the Workshop on Goal-Conditioned Reinforcement Learning (GCRL) at the Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[55] B. Yalcinkaya, N. Lauffer, M. Vazquez-Chanlatte, and S. Seshia. Compositional Automata Embeddings for Goal-Conditioned Reinforcement Learning. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[56] J. Zhang, J. Lehman, K. O. Stanley, and J. Clune. OMNI: Open-endedness via Models of human Notions of Interestingness. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

A Benchmark Details

In this section, we describe additional information about the benchmark introduced in Section 3. We organize the section as follows: Appendix A.1 describes some implementation details and features omitted in the main paper, and explains how HRMs are sampled via random walks, and Appendix A.2 provides some details about the structure of the problems in the hand-designed set. Illustrative example problems are given throughout the paper and the appendix (e.g., see Sections 4 and 5 and Appendices D and E.6).

A.1 Implementation Details

The benchmark is implemented in JAX [7] to enable seamless execution across CPUs and hardware accelerators (GPUs, TPUs). The implementation of Minigrid is based on that by Nikulin et al. [38].

Alphabet. Following the work of Mu et al. [36], we do not encode propositions for commands involving objects relative to the agent's position, e.g. *"go to a ball on your left"*. None of the commands imply the uniqueness of an object, e.g. *"pick up the yellow square"*. The *alphabet size* consists of 889 propositions, which is almost two orders of magnitude larger than the alphabets considered in previous work [22, 29, 52, 55]. The alphabet does not contain symmetric propositions (e.g., only contains one of `next_ball_key` and `next_key_ball`) and is derived under the assumption that two doors cannot be next to each other (one of the objects in a `next` proposition must be movable).

Task Sampling. For both the sequential and the random walk-based HRM samplers, there is a single edge between each pair of states. We provide an in-depth description of the random walk-based sampler in Appendix A.1.

Problem Sampling. We implement a *task-conditioned* sampler that complements the independent and level-conditioned sampling strategies (see Section 3.3). The task-conditioned sampler generates levels conditioned on the sampled HRM's propositions, ensuring that objects related to the HRM's propositions are present in the level. For example, given the proposition `next_ball_blue_square`, the level will be guaranteed to contain a blue ball and a square of any color. Upon choosing a conditioning strategy for the ablations (see Section 5.3), we selected the level-conditioned sampler as it better reflects real-world problem specification. In practice, the physical environment determines the agent's affordances—what actions and interactions are possible—which then constrains the meaningful range of tasks that can be specified. This mirrors how humans naturally assign tasks based on environmental context and available objects.

Random Walk-Based HRM Generation We here describe the method for sampling HRM tasks via random walks briefly outlined in Section 3. Figure 7 illustrates an HRM sampled via random walks. Unlike other hierarchies depicted throughout the paper, these HRMs may consist of multiple RMs, and each constituent RM need not be sequential. We specify HRMs using the following components, where m is the number of RMs in the hierarchy:

- a hierarchical graph structure $T = \langle \mathcal{V}_T, \mathcal{E}_T \rangle$, where $\mathcal{V}_T$ is a set of nodes for each constituent RM, and $\mathcal{E}_T \subseteq \mathcal{V}_T \times \mathcal{V}_T$ is a set of edges between these nodes;
- local RM graph structures $G_i = \langle \mathcal{V}_i, \mathcal{E}_i \rangle$ for $i \in [1, m]$, where $\mathcal{V}_i$ is a set of nodes for each RM state, and $\mathcal{E}_i \subseteq \mathcal{V}_i \times \mathcal{V}_i$ is a set of edges between these nodes;
- the call-labeling function $\mathcal{C} : \mathcal{E}_i \to \{G_j\}_{j=1}^m$, mapping each edge in RM i to an RM graph; and
- the edge-labeling function $\mathcal{P} : \mathcal{E}_i \to P$, mapping each edge in RM i to a proposition.

This specification can be easily transformed into the HRMs defined by Furelos-Blanco et al. [19]. In the following paragraphs, we describe how each of these components is sampled.

Hierarchical Graph Structure Sampling. The hierarchical graph structure T determines which RMs may be invoked from a given RM. Since the number of RMs in the hierarchy is m, the number of non-leaf nodes (i.e., RM nodes calling at least one RM) is at most $\lfloor m/2 \rfloor$. To generate the tree structure, we sample from a categorical distribution $\text{Cat}(p(\lfloor m/2 \rfloor), q)$ over the ordered integer partitions $p(\lfloor m/2 \rfloor)$, with associated categorical weights q. Here, *ordered* means that the order of summands matters. The value of the i-th summand in a partition specifies the number of children of the i-th node. Edges are connected sequentially based on these values. For balanced hierarchies, we recommend using a uniform distribution over the partitions, ensuring that all nodes have equal branching factors. Alternatively, the weights q can be tuned to bias the hierarchy toward flatter or deeper structures.

RM Sampling. The RM sampling is parallelized. Each RM i is instantiated via a realization of a Markov chain defined by a transition matrix $\mathbf{M}_i \in \mathbb{R}^{|U| \times |U|}$. For sequential RMs, $\mathbf{M}_i$ is a diagonal matrix with a unit diagonal shift. For directed acyclic RMs, $\mathbf{M}_i$ is lower triangular with a diagonal shift of one. Each matrix $\mathbf{M}_i$ is sampled as $\mathbf{M}_i \sim \mathcal{D}$, where $\mathcal{D}$ is typically chosen as a uniform distribution to avoid introducing bias, though it can be tuned to induce other desirable structures. The sampled matrix $\mathbf{M}_i$ is then normalized to be right-stochastic. Let $\bar{k}$ denote the average node connectivity in the sampled graph. Given $|U|$ as the number of states, the RM structure G_i is generated from the connectivity distribution induced by a random walk of length $\bar{k}|U|$, i.e.,

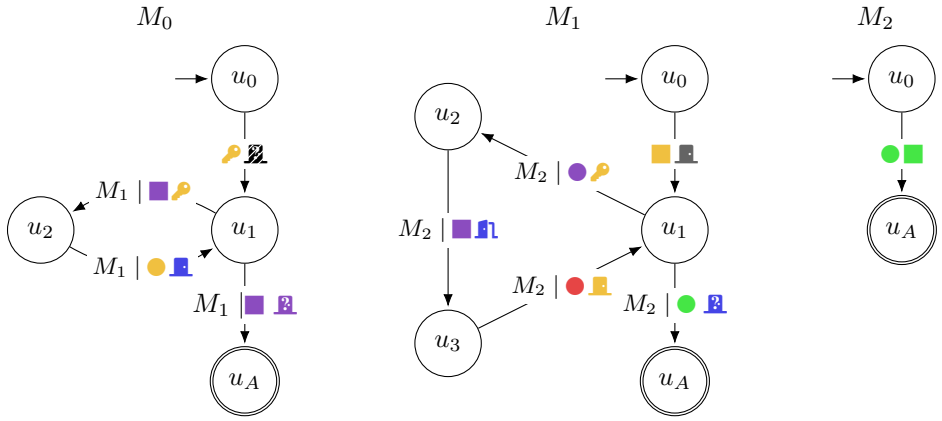


Figure 7: Example of an HRM generated via the random-walk based sampler. Symbols follow the descriptions in Figure 4. Calls to other RMs are denoted with $M_i \mid \varphi$, where M_i is the called RM and φ a formula that must be satisfied for the call to be started. Negative propositions in the cases where there are multiple outgoing transitions from a given state are omitted, e.g. the edges from u_1 in M_0 are actually labeled with the formulas ■🔑 ∧ ¬■ ? and ■ ? ∧ ¬■🔑.

$G_i \sim \mathrm{RW}_{\bar{k}|U|}(\mathbf{M}_i)$. After sampling a structure G_i, the random walk may be restarted with a fixed probability, and the final graph is obtained by taking the union of edges from all sampled trajectories. This mechanism allows control over the distribution of paths per RM. In the case of cyclic graphs, it imposes a lower bound on the number of distinct paths. For non-root RMs, we introduce one modification: we set $\mathbf{M}_i(0, j) = 0$ for all j, ensuring that the initial state has exactly one outgoing edge. The motivation for this constraint will become clear in the following sections.

Proposition Labeling. Labeling must be performed conditionally at random to avoid pathological cases. There are two such undesirable cases. First, a labeling is *tautological* if an incoming transition is labeled with a proposition p_k such that $p_k \to p_l \equiv \top$, where p_l is the label of an outgoing transition. Second, a labeling is *non-deterministic* if two outgoing transitions from the same node are labeled with propositions p_{l_1} and p_{l_2} such that either $p_{l_1} \to p_{l_2} \equiv \top$ or $p_{l_1} \leftarrow p_{l_2} \equiv \top$. For every node, each of the outgoing l transition labelings is represented as a one-hot encoded vector $\mathbf{p}_{\text{out}}^{(l)} \in \{0,1\}^P$ over the proposition set sampled from the conditional categorical distribution:

$$\mathbf{p}_{\text{out}}^{(l)} \sim \text{Cat}\left(P, \left(\prod_{i=1}^{k} \mathbf{C} \cdot \mathbf{p}_{\text{in}}^{(i)}\right) \odot \left(\prod_{i=1}^{l-1} \mathbf{C} \cdot \mathbf{p}_{\text{out}}^{(i)}\right) \odot \boldsymbol{\rho}_{\text{prop}}\right),$$

where $\mathbf{C}$ is a binary constraint matrix encoding the undesirable cases above and defined by

$$\mathbf{C}_{ij} = \begin{cases} 0 & \text{if } p_j \to p_i \equiv \top, \\ 1 & \text{otherwise,} \end{cases}$$

k is the number of incoming transitions, $\mathbf{p}_{\text{in}}^{(i)} \in \{0,1\}^P$ is a one-hot vector representing the labeling of the i-th incoming transition, and $\boldsymbol{\rho}_{\text{prop}} \in \{0,1\}^P$ is a categorical prior over the proposition set. To avoid introducing bias into the labeling, labels are sequentially assigned by sampling uniformly at random from the available outgoing edges $\mathcal{E}_i$. Finally, to ensure determinism, each label must be conjoined with the negation of its neighboring labels (i.e., labels on the transitions from the same node).

Call Labeling. Call labeling follows principles similar to those used for proposition labeling. However, in this case, the labels correspond to calls determined during hierarchy sampling; that is, the hierarchy defines the call alphabet for each RM. As with proposition labels, call labels are subject to consistency constraints. Since any call condition is conjoined with the initial transition labeling of the RM being called, the two must not be contradictory. This dependency also motivates the design choice of restricting RMs in the hierarchy to have exactly one initial transition without any call label. This restriction ensures tractability over call labelings. Let $\mathbf{K}$ be a binary compatibility matrix defined

by:

$$\mathbf{K}_{ij} = \begin{cases} 0 & \text{if } p_j \wedge p_i \equiv \bot, \\ 1 & \text{otherwise,} \end{cases}$$

where p_i and p_j are propositions. Then, for each edge $\mathcal{E}_i^{(j)}$, the call label c_j is sampled from the conditional categorical distribution:

$$c_j \sim \text{Cat}\left(C_i \cup \{\top\},\ \left(\mathbf{K} \cdot \mathbf{P}_{\text{init}}^{(i)}\right) \cdot \boldsymbol{\rho}_{\text{call}}^{(i)}\right),$$

where $C_i \subset \{G_j\}_{j=1}^m$ is the set of callable RMs as defined by the hierarchy T from RM i, and $\mathbf{P}_{\text{init}}^{(i)}$ is a matrix whose columns correspond to the proposition vectors associated with the initial transitions of the RMs in C_i. The vector $\boldsymbol{\rho}_{\text{call}}^{(i)}$ specifies the categorical prior over call probabilities.

A.2 Hand-Designed Evaluation Set

The hand-designed evaluation set consists of 150 problems. *Levels* mostly follow the grid morphologies used for training (see Appendix E.2), which divide the grid into different equally-sized rooms. Unlike random generation, we distribute objects across the grid in challenging ways, e.g. sparsely across different rooms, surrounded by several distractor objects, or enforcing door unlocking to reach certain areas. A small fraction of the problems (6) consist of levels without the specified room morphology. *Tasks* are all non-hierarchical RMs, most of which are *sequential*, following our default training scenario; however, some consist of multiple more than one path from u_0 to u_A, or longer paths than the training default (5). The RMs are designed to assess performance in scenarios where considering further ahead than the current subgoal (i.e., the literals labeling an outgoing edge from the current RM sate) is needed, or where the tasks are fully specified (i.e., contain all required subgoals) or underspecified (i.e., omit some subgoals, such as picking up a key to open a door). Examples of these hand-designed problems are shortly described in Section 5.2.

B Extended Related Work

In this section, we provide additional details on the related work covered in Section 6.

B.1 Unsupervised Environment Design (UED)

Adaptive UED methods often rely on regret estimates to determine the utility of levels. In this work, we employed MaxMC [23]; however, there are other methods such as the *positive value loss* [PVL; 23], which we experiment with in Appendix E.9. Both MaxMC and PVL rely on a single student. In contrast, the seminal UED approach, PAIRED [16], computes regret as the value difference between two student policies; specifically, the teacher aims to exploit the weaknesses of one student (the protagonist) in relation to the other student (the antagonist), producing an emergent curriculum of levels at the frontier of the protagonist's capabilities. We highlight some known limitations of regret estimation in Section 6.

PAIRED, as highlighted by Jiang et al. [23], is slow to adapt to changes in student policies since the teacher adapts through gradient updates. PLR [24] takes a different approach: the teacher does not generate levels, but curates a buffer of previously seen levels. Crucially, the teacher adapts faster than in PAIRED since it only implements a scoring mechanism. PLR$^\perp$ [23] extend PLR by training from buffer levels only, which improves the convergence to Nash equilibria. ACCEL [40] extends PLR methods by mutating previously sampled high-regret levels, showing that complexity can emerge starting from simple levels. Recent directions explore generative models for environment generation [13] and distribution shift in adaptive curricula [20].

Unlike previous UED approaches, ATLAS does not assume a fixed task and induces autocurricula over both tasks and levels. Furthermore, it implements structure-aware task mutations that enable compounding complexity from very simple tasks (one transition RMs) and levels.

Table 1: Comparison of problem-conditioned RL methods using formal task representations.

Algorithm	Conditioning	Curriculum Target	Autocurricula	Alphabet Size	Sampling	Invalid Problems
Kuo et al. [29]	LTL/Network-Operators	LTL complexity	✗	≤ 9	DR	✗
Vaezipoor et al. [52]	LTL syntax tree	No curriculum	✗	≤ 12	DR	✗
Qiu et al. [43]	Propositions	No curriculum	✗	≤ 16	DR (prop.)	✗
Yalcinkaya et al. [54]	DFA	No curriculum	✗	3	DR	✗
Yalcinkaya et al. [55]	cDFA	No curriculum	✗	≤ 12	DR (RAD)	✗
Jackermeier and Abate [22]	Reach-avoid sequences	LTL complexity	✗	≤ 20	DR	✗
ATLAS (ours)	RMs	RM & level complexity	✓	≤ 889	PLR, ACCEL	✓

B.2 Formal Language Conditioning

We provide further details on the core components of the methods that learn policies conditioned by formal language task specifications, mainly LTL and automata. Table 1 provides an overview of the main distinguishing factor, which we elaborate on in the following paragraphs and Section 6.

Sampling and Mutations. *LTL-conditioned* methods [e.g., 22, 29, 52] typically generate tasks by procedurally sampling LTL formulas combining logical and temporal operators from a context-free grammar. These methods build rules into the grammar that prevent logically unsatisfiable tasks. Qiu et al. [43] propose a method that trains from randomly sampled propositions instead of LTL tasks—the policies are proposition-conditioned and then ensembled together to satisfy LTL specifications.

In the context of *automata-conditioned* methods, Yalcinkaya et al. [54] sample deterministic finite automata (DFA) representing reach-avoid tasks. The path length is specified within a range similar to ours (3–7). Yalcinkaya et al. [55] propose a family of DFAs called *reach-avoid derived* (RAD). The authors pre-train DFA encoders on this class and demonstrate zero-shot generalization to other DFAs. The key insight is that any path through a DFA can be seen as a series of reach-avoid sub-tasks. RAD DFAs are generated through mutations from reach-avoid DFAs. Each mutation involves randomly changing a transition, converting the accepting state into a sink, and minimizing the resulting DFA to ensure simplicity. A filtering step rejects resulting trivial, uninteresting tasks (e.g., single-state DFAs); hence the training distribution consists of non-trivial, solvable tasks. ATLAS does not perform the mutations on random samples, but on high-regret ones; further, it indirectly performs the filtering through regret scoring. Our random walk-based sampling strategy acts as an alternative to performing mutations on reach-avoid DFAs, still permitting the application of mutations over the samples. The authors also introduce cDFAs, a parallel composition of two DFAs that enables executing the DFAs in an interleaved manner. HRMs, the formalism employed in ATLAS, composes automata sequentially at arbitrarily many hierarchical levels rather than in parallel. Previous work has considered parallel compositions of RMs in the multi-agent setting [37].

There are two key differences between ATLAS and previous work. First, the problems tackled by previous LTL- and automata-conditioned methods often consist of small propositional alphabets, with the maximum being 20 in the work by Jackermeier and Abate [22]; in contrast, ATLAS employs an alphabet of 889 propositions. Second, as mentioned in Section 6, prior work employs levels where any proposition in the alphabet can be observed. For example, the commonly used LETTERWORLD domain contains 12 letters in a 7×7 grid, each corresponding to a proposition and appearing twice. Similarly, the also typical ZONES domain determines four colors (i.e., four propositions) and any level has two regions for each of these colors. On the other hand, the problems we consider do not guarantee that the objects mentioned in an RM will appear or be reachable.

Curriculum. The LTL-conditioned methods proposed by Kuo et al. [29] and Jackermeier and Abate [22] use a handcrafted curriculum that starts with simple tasks and gradually progresses to more complex tasks (e.g., longer formulas) as satisfactory performance is achieved. Similar curricula, combined with a small predefined set of tasks, have been previously been used with policy sketches [3], neural programs [41], and hierarchies of reward machines [19]. In contrast, ATLAS leverages regret-based UED to generate autocurricula of solvable yet challenging problems; further, unlike previous approaches, ATLAS performs curricula over both tasks and levels rather than tasks alone.

C Architecture Details

The following paragraphs provide details on the problem-conditioned architecture introduced in Section 4.1.

C.1 Environment Observation Encoding

In the Minigrid implementation by Nikulin et al. [38], observations are $V \times V \times 2$, where $V = 5$ is the size of the agent's field of view, the first dimension contains the object identifiers, and the second dimension contains the colors. If the agent is carrying an object, the object appears in the agent's position.

Following the example implementation by Nikulin et al. [38], the object and color identifiers in the observation are separately embedded into 16-dimensional vectors. The object and color embeddings are concatenated. The resulting $V \times V \times 32$ tensor is processed by a 3-layer convolutional neural network (CNN) with ReLU activations and 32, 64, and 64 filters, respectively. All kernels are 2×2 with a stride of 1 and zero-padding.

C.2 Reward Machine Encoding

Given an RM $M = \langle U, P, \delta, \mathcal{R}, u_0, u_A \rangle$ and one of its states $u \in U$, the encoding is performed with a *graph neural network* (GNN). The section is organized as follows. First, we describe the GNN architecture we use. Second, we describe how the graph passed to the GNN is constructed from the input RM. Finally, we describe the values of the GNN-related hyperparameters. Our *implementation* is based on that from the `jraph` library [21].

Graph Convolutional Networks [GCNs; 27]. Given a graph $G = \langle \mathcal{V}, \mathcal{E} \rangle$ with nodes $v_i \in \mathcal{V}$ and edges $\langle v_i, v_j \rangle \in \mathcal{E}$, GCNs update node representations $\mathbf{h}_i \in \mathbb{R}^n$ by aggregating those from neighboring nodes. Each node v_i is randomly initialized with features $\mathbf{h}_i^{(0)} \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$, where $\sigma^2 = 0.1$ [1, 31]. Our GCN updates the node features through L layers of message passing:

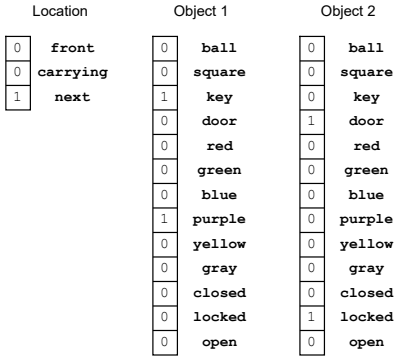
$$\mathbf{h}_i^{(l+1)} = \sigma\left(\text{LN}^{(l)}\left(\mathbf{W}_{\text{self}}^{(l)}\mathbf{h}_i^{(l)} + \mathbf{W}_{\text{neigh}}^{(l)} \sum_{j \in \mathfrak{N}(i)} \frac{\mathbf{h}_j^{(l)} \oplus \mathbf{e}_{j,i}}{|\mathfrak{N}(i)|}\right)\right),$$

where $\mathfrak{N}(i)$ denotes the neighbors of node v_i, $\mathbf{W}_{\text{self}}^{(l)} \in \mathbb{R}^{n \times n}$ and $\mathbf{W}_{\text{neigh}}^{(l)} \in \mathbb{R}^{n \times (n+m)}$ are learnable parameters, $\mathbf{e}_{j,i} \in \mathbb{R}^m$ encodes the formula of edge $\langle v_j, v_i \rangle$, σ is an activation function, LN denotes layer normalization [5], and $\oplus$ denotes concatenation. The parameters $\mathbf{W}_{\text{self}}^{(l)}$ are used to capture the same node's features in the previous layer, while $\mathbf{W}_{\text{neigh}}^{(l)}$ are used for the neighboring nodes. We separate them since RM states do not have self-loops labeled by formulas. R-GCNs [46] handle this similarly by having a set of parameters for different relation types and specifying self-loops as a type of relation. The resulting features $\mathbf{h}_i^{(L)}$ undergo a last linear transformation $\mathbf{W}_{\text{last}} \in \mathbb{R}^{n \times n}$.

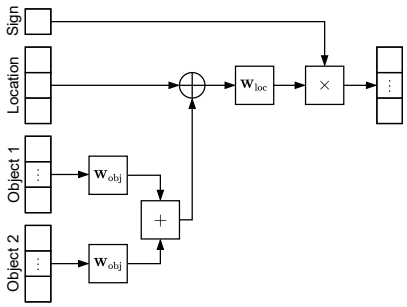
Graph Construction from a Reward Machine. The *input graph* is constructed from an RM by reversing the edges, enabling each state to be contextualized by future states reachable within L transitions. This design enables considering what needs to be accomplished far into the future, rather than being myopic (i.e., looking one step ahead). Previous approaches for conditioning policies on LTL formulas [e.g., 52] and finite-state machines [e.g., 54, 55] follow the same reasoning.

The *edge features* $\mathbf{e}_{j,i}$ are built by (i) embedding each literal (proposition or its negation) of the formula, (ii) aggregating the literal embeddings by summing them, and (iii) applying a linear transformation $\mathbf{W}_{\text{lit}} \in \mathbb{R}^{m \times m}$. We consider two methods for constructing the *literal embeddings*:

- *Domain Independent.* Literals are embedded independently through an embedding matrix with $2N + 1$ rows, where N is the alphabet size. Negative literals correspond to rows $0, \ldots, N - 1$; $\top$ (the value *true*) corresponds to the N-th row; and positive literals correspond to rows $N + 1, \ldots, 2N$.
- *Domain Dependent.* The proposition's location and object information is decomposed, embedded separately, and aggregated. Figure 8 illustrates and describes how each literal embeddings are constructed.



(a) Encoding decomposition for the proposition `next_key_purple_door_locked`.



(b) Construction of a literal embedding from the decomposed proposition encoding.

Figure 8: Phases in the literal embedding construction. **(a)** A binary encoding of the proposition is first constructed. **(b)** The literal embedding is built using the proposition decomposition. First, the binary encodings of the objects undergo a linear transformation $\mathbf{W}_{\text{obj}} \in \mathbb{R}^{m \times 13}$ and their resulting representations are aggregated using an order invariant operation (sum). Second, the location binary encoded is appended to the object representation and linearly transformed with $\mathbf{W}_{\text{loc}} \in \mathbb{R}^{m \times (m+3)}$. Finally, the sign of the literal (1 if positive, -1 if negative) is applied to produce the final embedding.

In both cases, the literal embeddings are learnable.

Hyperparameters. The GCN consists of $L = 5$ layers, chosen after the maximum number of transitions from the initial state u_0 to the accepting state u_A that can be sampled in our training setting. The number of node features is $n = 128$ and the number of edge features is $m = 64$. The activation functions are ReLUs.

C.3 Encoding Aggregation and Actor-Critic Heads

Given the encodings for the current observation and the current RM state, we concatenate them with a 16-dimensional embedding of the previous action. The resulting embedding is processed by a gated recurrent unit [GRU; 12] producing 512 features. RMs implicitly encode a history over propositions, so the application of the GRU over the RM state encoding could be deemed unnecessary; however, we found that it experimentally works better.

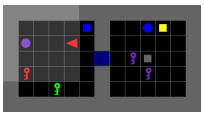
The embedding—aggregating the observation, RM state, and action information—is then processed by the actor and critic heads, each consisting of two hidden layers with 256 rectifier units. The actor's output layer produces a logit for each action, while the critic's output layer produces a scalar estimating the value for the input observation, RM state, and action.

D Mutation Details

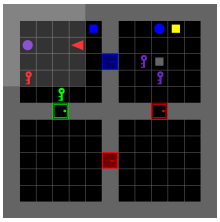
In this section, we describe the preconditions and effects of each edit introduced in Section 4.2. These edits are conceptually applicable to any problem, i.e. task-level pair. However, we focus on our main *training* setting where (i) RM tasks are sequential with at most 6 states, and (ii) levels are structured into $\{1, 2, 4, 6\}$ rooms of size 7×7. One-room levels are 7×7 and contain 1–5 objects, two-room levels are 7×13 and contain 1–10 objects, four-room levels are 13×13 and contain 4–15 objects, and six-room levels are 13×19 and contain 7–20 objects. The minimum number of objects is determined as the maximum between 1 and the number of doors in the level.

D.1 Level Edits

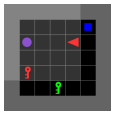
There are eight types of level edits, illustrated in Figure 9:



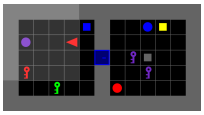
(a) Source level



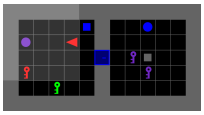
(b) AddRooms



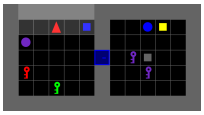
(c) RemoveRooms



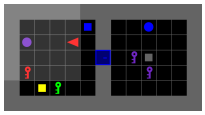
(d) AddObject



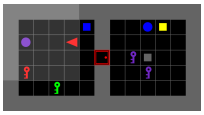
(e) RemoveObject



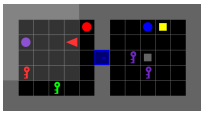
(f) MoveAgent



(g) MoveObject



(h) ReplaceDoor



(i) ReplaceNonDoor

Figure 9: Level edit examples.

AddRooms Applicable if the source level has 1, 2, or 4 rooms. Transforms one-room levels into two-room levels, two-room levels into four-room levels, and four-room levels into six-room levels. The extended side of the source level is determined uniformly at random—that is, given a one-room level, a room is added to the left or the right; given a two-room level, two rooms are added above or below; and given a six-room level, two rooms are added to the left or the right.

RemoveRooms Applicable if the source level has 2, 4, or 6 rooms. Transforms levels following the inverse order of AddRooms. The removed rooms cannot have the agent in them—given a two-room level, remove a room; given a four-room level, remove a row of rooms; and given a six-room level, remove the leftmost or rightmost column (if the agent is in the center column, choose one randomly).

AddObject Applicable if the source level does not contain the maximum number of objects allowed (room-dependent, see above). A random non-door object (type and color) is determined and placed on a random free position.

RemoveObject Applicable if the source level does not contain the minimum number of objects. A random non-door object on the grid is removed.

MoveAgent Applicable if there are free locations, which is guaranteed by the maximum number of objects being lower than the number of locations per level. The agent is moved to a random free location with a randomly determined orientation.

MoveObject Applicable if there are free locations. An existing non-door object is randomly chosen and moved to a random free location.

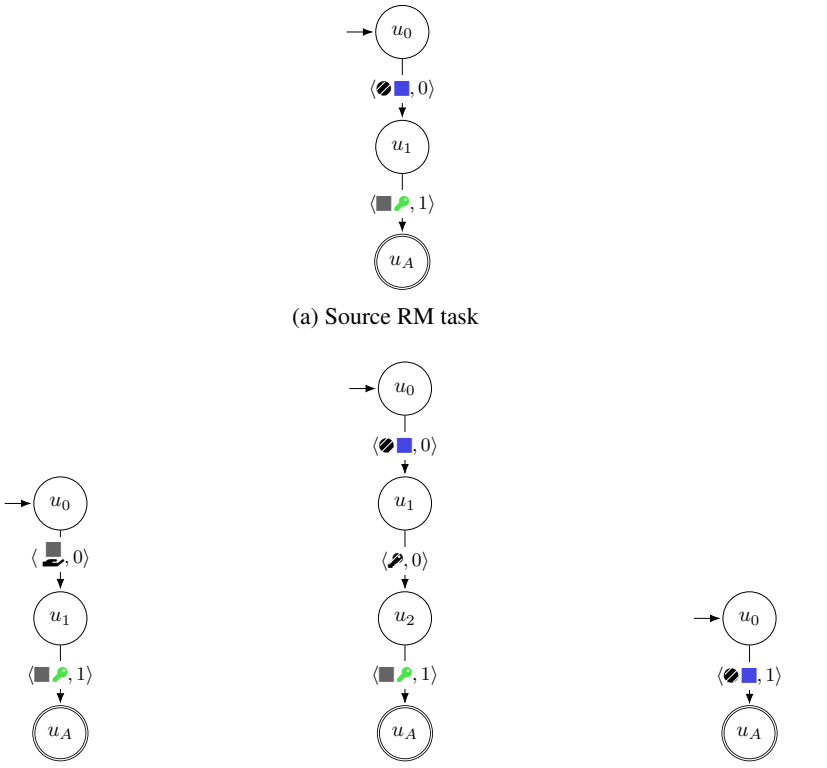
ReplaceDoor Applicable if the level contains doors (i.e., more than one room). An existing door is randomly chosen and replaced with a new door whose color and status (locked, closed, open) are randomly determined.

ReplaceNonDoor Applicable if the level contains non-door objects. An existing non-door object is randomly chosen and replaced with a new random non-door object.

D.2 Task Edits

There are three types of task edits, illustrated in Figure 10:

SwitchProposition Always applicable. An edge is randomly selected, and its proposition is replaced by another one chosen uniformly at random from the alphabet.



(a) Source RM task

(b) SWITCHPROPOSITION (c) ADDSTATE (d) REMOVESTATE

Figure 10: Task edit examples. Zero-reward self-transitions are omitted for simplicity.

ADDSTATE Applicable if the number of states is less than the maximum. Adds a new state at the start, middle, or end of the existing state sequence. If inserted in the middle, transitions are rewired to maintain connectivity. The outgoing transition from the new state is labeled with a proposition chosen uniformly at random from the whole alphabet.

REMOVESTATE Applicable if there are more than two states. Removes one of the states and, if it is a non-accepting state, its outgoing transition as well. If the removed state is the initial state, the state pointed to by it becomes the new initial state. If the removed state is the accepting state, the state that pointed to it becomes the new accepting state.

The reward transition function of the resulting RM is adjusted to make it *sparse* (reward of 1 in transitions to u_A), which is the training default (see Section 5.1).

D.3 Hindsight Edits

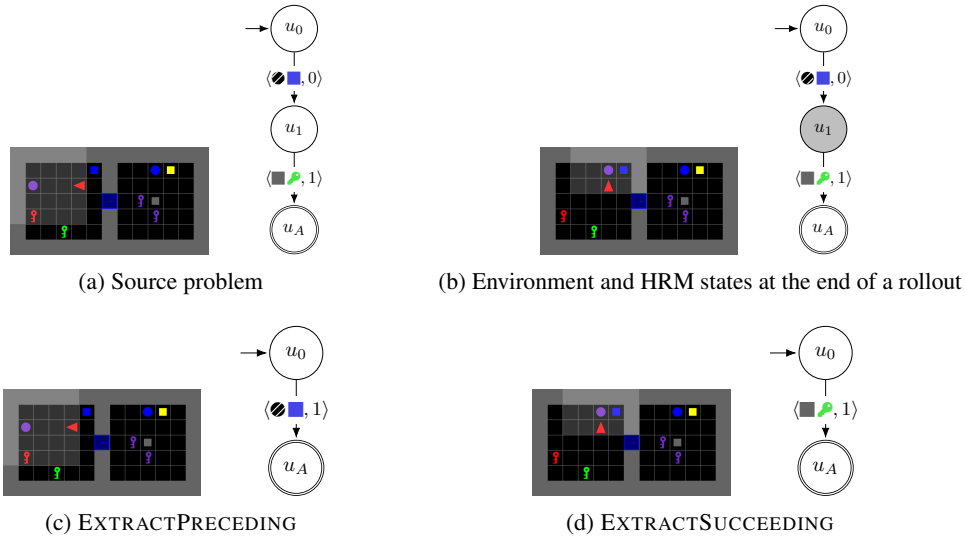
These edits are applicable if the last RM state $u \in U$ in a rollout is not the initial or the accepting state. Further, they can only be selected as the first step of the edit sequence. There are two edit types, illustrated in Figure 11, both of which jointly modify the level and the task:

EXTRACTPRECEDING Keeps the original level and derives a new RM with u as the accepting state.

EXTRACTSUCCEEDING Derives a new level from the last environment state and a new RM with u as the initial state.

E Experimental Details

In this section, we explain our experimental setup, hyperparameters, and additional results for the experiments described in Section 5. Our codebase is fully implemented in JAX [7]. We extended the JaxUED [15], a collection of UED algorithm implementations in JAX, to support problems (levels



(a) Source problem

(b) Environment and HRM states at the end of a rollout

(c) ExtractPreceding

(d) ExtractSucceeding

Figure 11: Hindsight edit examples.

and tasks) rather than only levels. Our implementation of Minigrid is based on that by Nikulin et al. [38].

The experiments were executed across three different Linux clusters, respectively consisting of (i) NVIDIA RTX 6000 Ada Generation GPUs and Intel Xeon Gold 6348 CPUs, (ii) NVIDIA L40S GPUs and Intel Xeon Platinum 8358 CPUs, and (iii) NVIDIA A100-SXM4-80GB GPUs and Intel Xeon Platinum 8562 CPUs. For all experiments we reserved 32GB of RAM. Each experiment takes approximately 40 hours to complete; however, the time can be substantially reduced by logging checkpoints and results less often. Our results report aggregate performance across 5 seeds.

E.1 Hyperparameters

Table 2 shows the table containing the RL-related hyperparameters used in the final experiments, for which we use the Adam optimizer [26]. We refer the reader to Appendix C for the architectural details.

To determine the final hyperparameter values, we sampled a random level-conditioned problem set. The problems underwent a solvability check, which we describe in Appendix E.3. The RMs in the set are all sequential. The problem set is constituted by 300 samples that result from taking 5 samples for each combination of: number of rooms (1, 2, 4, 6), number of objects and number of transitions (1–5). For each number of rooms, we distinguish three object intervals:

- One room: 1–2, 3–4, 5.
- Two rooms: 1–3, 4–7, 8–10.
- Four rooms: 4–7, 8–11, 12–15.
- Six rooms: 7–10, 11–16, 17–20.

This set is exclusively used for validation.

The hyperparameter sweep was as follows. We performed a grid search for PLR$^{\perp}$ across the buffer capacity $\{25000, 50000\}$, the replay probability $\{0.5, 0.9\}$, the PPO entropy loss coefficient $\{0.0, 0.001, 0.01\}$, and whether to feed the RM embedding into the RNN together with the observation embedding. For ACCEL, we fixed a replay probability of 0.9, a buffer size of 50000, and a maximum number of edits of 10, and repeated the same grid search over the same remaining hyperparameters while sweeping over the minimum number of edits $\{3, 5, 7, 10\}$. In hyperparameter searches preceding the final one, we swept also over the staleness coefficient $\{0.1, 0.3\}$, discount rate $\{0.9, 0.95, 0.98, 0.99\}$, GAE lambda $\{0.9, 0.95, 0.98\}$, sampling temperature $\{0.3, 1.0\}$, learning rate $\{1 \times 10^{-5}, 5 \times 10^{-5}\}$, and value loss coefficient $\{0.5, 0.75\}$.

Table 2: List of environment, PPO, PLR$^{\perp}$ and ACCEL hyperparameters.

Parameter	Value
Environment	
Maximum episode length	512
PPO	
# Environment steps (Seq.)	4,194,304,000
# Environment steps (DAG)	6,291,456,000
Discount rate γ	0.99
GAE λ	0.9
# Parallel environments	4,096
PPO rollout length	512
PPO epochs	4
PPO minibatches per epoch	128
PPO clip range	0.2
PPO max. gradient norm	0.5
Adam learning rate	5×10^{-5}
Adam ϵ	1×10^{-5}
Value loss coefficient	0.5
Entropy loss coefficient	0.01
PLR$^{\perp}$	
Buffer capacity	50,000
Prioritization	rank
Temperature	1.0
Replay rate	0.5
Staleness coefficient	0.1
Score function	MaxMC
ACCEL	
Replay rate (full)	0.9
Number of edits	$\mathcal{U}\{7, 10\}$
ACCEL-0	
Replay rate	0.99
Number of edits	$\mathcal{U}\{7, 10\}$

E.2 Experimental Setup Details

We describe some omitted details from the experimental setup description in Section 5.1.

Level Generation. Training levels (see Section 3.1) are generated as follows:

1. A *number of rooms* is uniformly sampled from $\{1, 2, 4, 6\}$. Rooms are 5×5, surrounded by walls (total dimension 7×7). Two-room levels are 7×13, four-room levels are 13×13, and six-room levels are 13×19. Levels with more than one room have doors between each pair of adjacent rooms, always in the middle of their dividing wall.
2. A *number of objects* is sampled uniformly from a grid-dependent range. One-room levels can have 1–5 objects, two-room levels can have 1–10 objects, four-room levels can have 4–15 objects, and six-room levels can have 7–20 objects. The minimum is determined by the number of doors.
3. *Non-door* objects are generated by choosing their type, color, and location uniformly at random. *Door* objects are analogously generated by choosing their state (open, closed, locked) and color.
4. The *agent* is randomly placed in a free position.

Metrics. The CVaR problem set consists of 10,000 samples, as recommended by Rutherford et al. [44], generated with the *level-conditioned* sampler described in Section 3.3. Sampling problems

Table 3: Percentage of solvable problems when the RMs are sequential and directed acyclic graphs.

Problem Sampling	Sequential	Directed Acyclic Graph
Independent	2.7 ± 0.3	3.9 ± 0.2
Level-Conditioned	83.4 ± 0.4	84.8 ± 0.5

conditionally on the level increases the solvability rate (see Appendix E.3); hence, although results do not substantially differ, the level-conditioned set covers a more diverse set of problems than if the level and the task were independently sampled. The problems are guaranteed to be *solvable* using the process described in Appendix E.3.

E.3 Fraction of Solvable Randomly Generated Problems

We analyze the fraction of solvable problems across batches sampled via the different sampling strategies. We report the average solvability and the standard deviation across 5 batches, where each batch consists of 4096 problems, as in our experiments.

To evaluate whether a problem is *solvable*, we decompose the constituent RM into paths to the accepting state and determine if the formulas along these paths are satisfiable. An RM is considered solvable if the formulas in at least one path are satisfiable. A formula is considered satisfiable if the objects associated with it are reachable by the agent. For example, the formula `front_ball` is satisfiable if there is a ball within the reach of the agent. Reachability is determined by locked doors—hence, if a given formula cannot be satisfied by the objects within reach, we select a locked door whose color matches a key within reach. This procedure derives a tree where each child node increases the reachability with respect to its parent. Maintaining a tree, keeping different orderings on the opening of the locked doors, is important because only some of them might guarantee solvability. For instance, if there is a single green locked door and the reachability procedure opens it, a subsequent formula `front_door_green_locked` will not be satisfiable.

Table 3 shows the results for the cases where the *task sampler* generates sequential and directed acyclic graphs, and the *problem sampler* generates levels and tasks independently, or conditionally on the level. The instantiation of the samplers is as explained in Section 5.1. We observe that independent sampling produces batches with barely any solvable problems (around 3–4%), whereas level-conditioned sampling produces a large majority of solvable problems (around 85%). As shown in Section 5.3, increasing the number of solvable problems per batch results in a sensible performance improvement for DR, which closes the gap with respect to PLR$^{\perp}$ and ACCEL.

Table 4 decomposes the results for the *sequential* RM sampler across: problem sampler (independent, conditioned), number of transitions (1–5), number of rooms (1, 2, 4, 6), and number of objects (L, M, H). The symbols for the number of objects denote different intervals (low, medium, high) depending on the number of rooms:

- One room: L (1–2), M (3–4), H (5).
- Two rooms: L (1–3), M (4–7), H (8–10).
- Four rooms: L (4–7), M (8–11), H (12–15).
- Six rooms: L (7–10), M (11–16), H (17–20).

Following the trend from Table 3, we observe that sampling conditionally on the level results in a substantially higher fraction of solvable problems. In the independent case, having fewer transitions and more rooms (and, hence, more objects) results in more solvable problems since there is a higher chance the transitions will refer to objects in the level. The percentage of solvable problems is extremely low, showing the potential of UED approaches in these settings. We emphasize that these results are illustrative and that none of these combinations are exclusively used in training the policies; instead, given a fixed problem sampling strategy (independent, conditioned), we determine the number of transitions, number of rooms, and number of objects uniformly at random.

E.4 Extended Main Results

We provide some more details on the core results described in Section 5.2.

Table 4: Percentage of solvable samples produced by the sequential sampler by fixing the problem sampler (independent, level-conditioned), number of transitions (1–5), number of rooms (1, 2, 4, 6), and a range for the number of objects (L, M, H).

		1			2			4			6		
		L	M	H	L	M	H	L	M	H	L	M	H
Independent	1	0.9 ± 0.1	2.6 ± 0.2	4.4 ± 0.3	2.2 ± 0.1	7.6 ± 0.3	12.1 ± 0.2	5.2 ± 0.3	14.6 ± 0.6	22.4 ± 0.3	6.7 ± 0.3	20.0 ± 0.4	30.9 ± 0.8
	2	0.0 ± 0.0	0.0 ± 0.0	0.2 ± 0.1	0.1 ± 0.0	0.6 ± 0.1	1.5 ± 0.3	0.3 ± 0.1	2.5 ± 0.2	5.6 ± 0.4	0.6 ± 0.2	4.4 ± 0.1	10.6 ± 0.3
	3	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.0	0.3 ± 0.0	0.0 ± 0.0	0.5 ± 0.1	1.4 ± 0.1	0.1 ± 0.0	1.2 ± 0.1	4.2 ± 0.2
	4	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.0	0.0 ± 0.0	0.1 ± 0.1	0.4 ± 0.1	0.0 ± 0.0	0.3 ± 0.0	1.5 ± 0.1
	5	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.1	0.0 ± 0.0	0.2 ± 0.1	0.6 ± 0.1
Conditioned	1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	90.5 ± 0.4	85.1 ± 0.5	86.6 ± 0.6	84.6 ± 0.6	84.0 ± 0.5	86.0 ± 0.4	82.3 ± 0.4	82.1 ± 0.6	85.8 ± 0.4
	2	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	87.9 ± 0.5	79.5 ± 0.7	80.2 ± 0.4	79.5 ± 0.7	78.4 ± 0.5	81.3 ± 0.7	76.7 ± 0.5	76.5 ± 0.6	80.9 ± 0.5
	3	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	87.0 ± 0.4	76.4 ± 0.8	76.8 ± 0.7	77.2 ± 0.7	76.1 ± 0.8	78.6 ± 0.6	73.8 ± 0.6	73.4 ± 0.5	78.5 ± 0.1
	4	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	86.5 ± 0.5	74.4 ± 0.7	74.6 ± 0.6	75.8 ± 0.8	74.1 ± 0.7	76.7 ± 0.6	72.0 ± 0.7	71.4 ± 0.4	76.8 ± 0.3
	5	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	86.1 ± 0.6	73.0 ± 0.5	73.2 ± 0.8	74.7 ± 0.4	72.4 ± 0.6	75.0 ± 0.6	70.8 ± 0.9	69.9 ± 0.5	75.1 ± 0.1

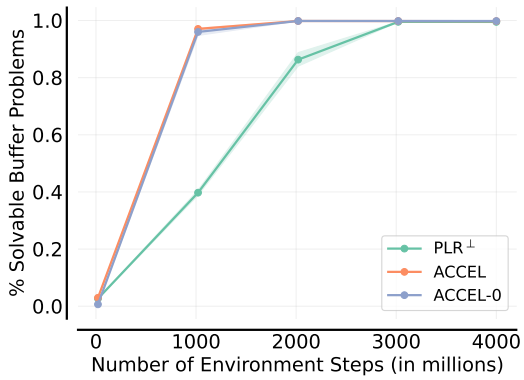


Figure 12: Fraction of solvable buffer problems throughout training in the *independent* sampling setting.

Performance. Figure 12 shows the percentage of solvable problems in the buffer throughout training. As training progresses, PLR$^{\perp}$, ACCEL, and ACCEL-0 curate a buffer mostly consisting of solvable problems. The performance for each individual hand-designed problem is reported in Appendix E.10.

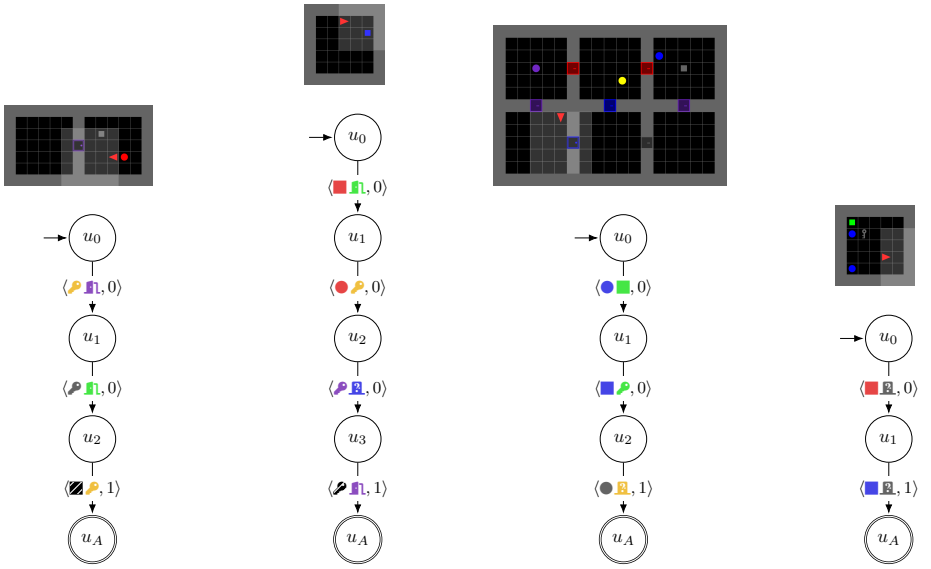
Curriculum Analysis. Figures 13 to 16 show a sequence of training problems for DR, PLR$^{\perp}$, ACCEL and ACCEL-0. In the case of the last three approaches, these problems are samples with high probability from the buffer, gradually become more complex, and are often solvable. In the case of DR, the generated problems are of arbitrary complexity and often unsolvable.

Mutation Analysis Figure 17a illustrates the evolution of mutations in buffer problems throughout training. Both ACCEL and ACCEL-0 exhibit similar trends. Hindsight edits are barely present (possibly) because they are hard to sample: they are only applicable if the RM state is not u_0 or u_A, and can only be randomly selected as the first edit in the sequence. Non-hindsight edits have a similar and continued presence in the buffer.

Figure 17b shows the average number of edits that produced each buffer problem from its parent. Non-mutated problems (i.e., problems with zero edits) are accounted for. In our experiments, the number of edits is sampled uniformly at random between 7 and 10. For both ACCEL variants, the average number of edits quickly grows close to the maximum, showing that (i) longer edit sequences are beneficial, and (ii) the buffer eventually consists mostly of mutated problems.

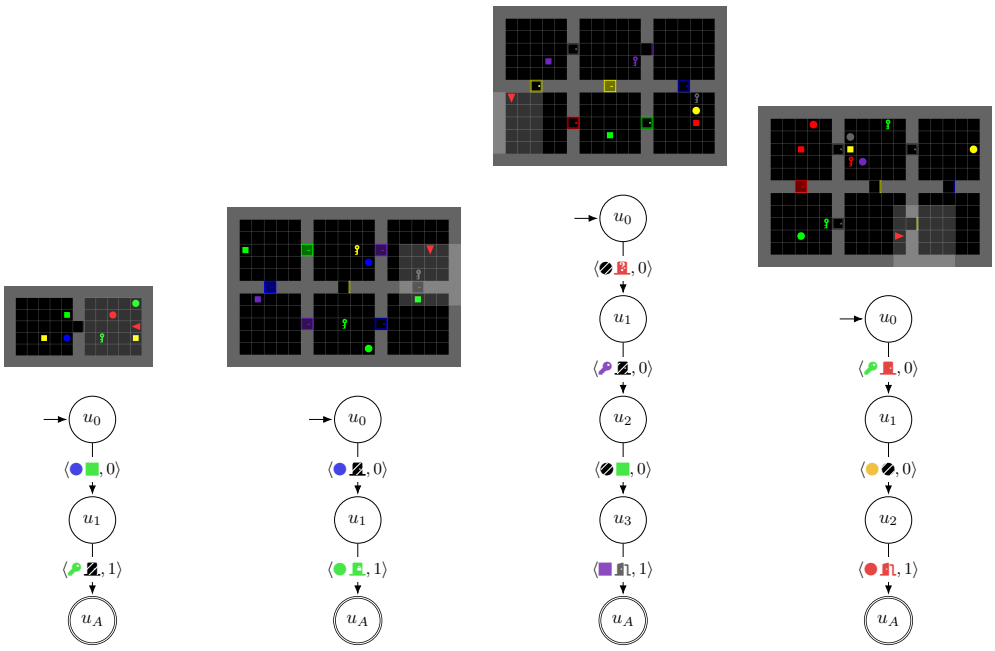
E.5 Extended Problem Sampling Ablation Results

Figures 18a and 18b show the performance of DR, PLR$^{\perp}$ and ACCEL in the *level-conditioned* setting, i.e. when RM task sampling is conditioned on the level (see Section 3). By sampling RMs conditionally, the fraction of solvable problems increases from 2.7% to 83.4%. DR is the approach that benefits most from the higher number of solvable problems per batch; indeed, it is competitive with PLR$^{\perp}$ and ACCEL in terms of robustness (CVaR) and generalization on the hand-designed test



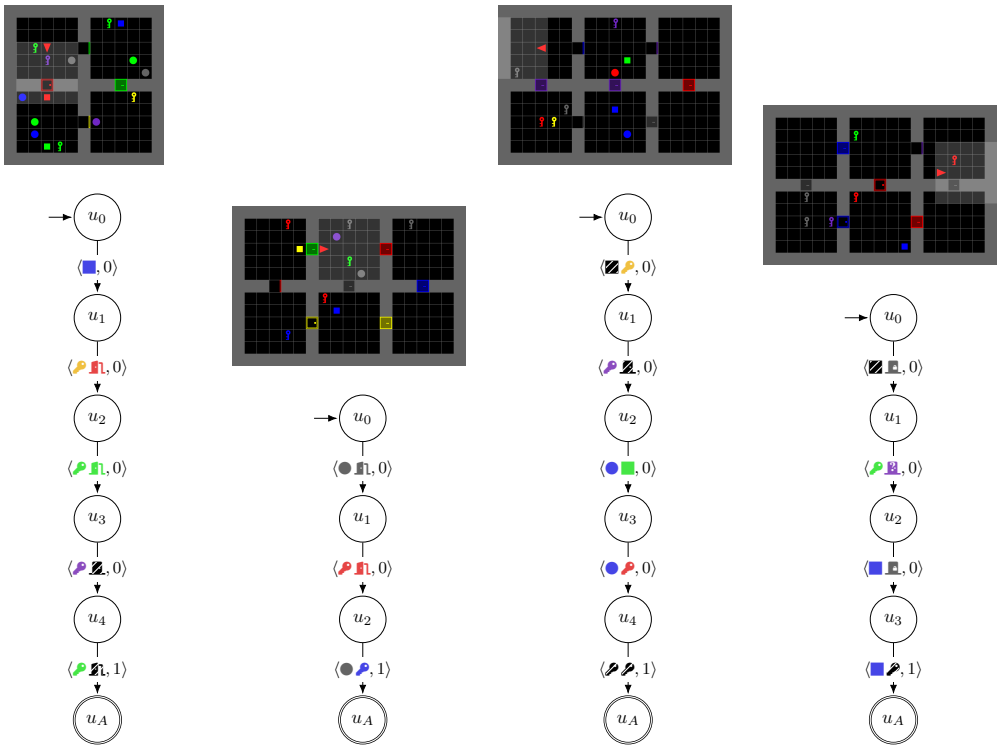
(a) 1×10^9 (b) 2×10^9 (c) 3×10^9 (d) 4×10^9

Figure 13: Samples of generated problems at different points in time (in number of environment steps) using DR.



(a) 1×10^9 (b) 2×10^9 (c) 3×10^9 (d) 4×10^9

Figure 14: Samples of generated problems at different points in time (in number of environment steps) using PLR$^{\perp}$.



(a) 1×10^9 (b) 2×10^9 (c) 3×10^9 (d) 4×10^9

Figure 15: Samples of generated problems at different points in time (in number of environment steps) using ACCEL.

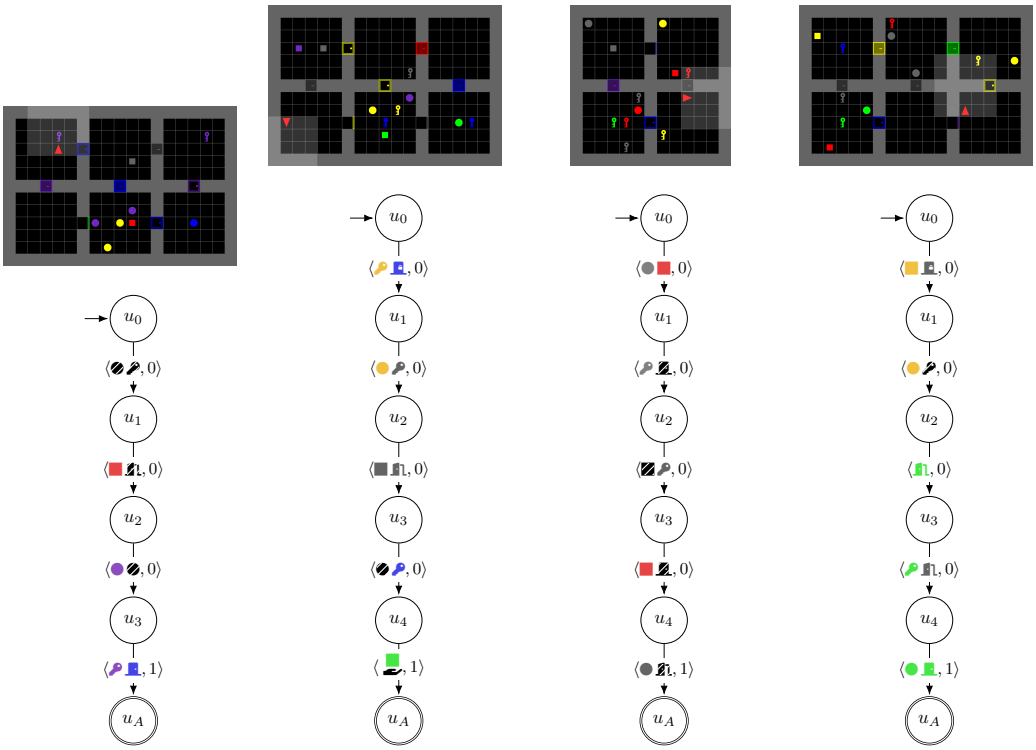
set. Figure 18c illustrates the test performance for independent and level-conditioned sampling at the end of training. Except for DR, all approaches perform similarly for both sampling methods, further emphasizing the robustness of PLR$^{\perp}$ and ACCEL in settings where solvable problems are difficult to sample.

Figure 19 illustrates the evolution of problems in the buffer throughout training. As for the independent sampling case (see Figure 5), we observe that PLR$^{\perp}$, ACCEL, and ACCEL-0 start prioritizing simple problems (few states, rooms, and objects) and progressively switch towards harder problems. The curriculum over the number of rooms and objects is similar for both independent and level-conditioned sampling. However, the curriculum over the number of states differs: converging toward sampling RMs with increasingly more states is more challenging in the independent sampling case. This is especially noticeable for PLR$^{\perp}$, which generates RMs with 3–4 states (resp. 5–6) for independent sampling (resp. level-conditioned) by the end of training. We hypothesize that enabling the sampling of a higher fraction of solvable problems due to level-conditioning induces this effect on PLR$^{\perp}$.

Figure 20 shows the fraction of solvable problems in the buffer throughout training. As with independent sampling (Figure 12, Appendix E.4), PLR$^{\perp}$ and ACCEL manage to curate a buffer mostly constituted by solvable problems. We emphasize that although the generator produces level-conditioned samples, the edits are not level-conditioned and may produce unsolvable problems.

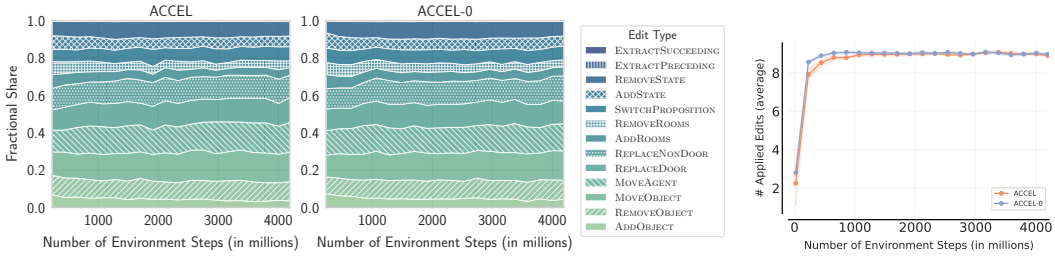
E.6 Extended Task Sampling Ablation Results

Figure 21 shows the performance of DR and PLR$^{\perp}$ in the setting where RM tasks are *directed acyclic graphs* (DAGs) generated with the *random walk-based* sampler. The maximum number of states is set to 6 (like for the sequential sampler), and the maximum number of paths from the initial state u_0 to the accepting state u_A is set to 2. We analyze the results with independent sampling and level-conditioned sampling. The CVaR of the solve rate shows that PLR$^{\perp}$ is far more robust than DR in the independent sampling setting; however, in the level-conditioned sampling setting, PLR$^{\perp}$ is



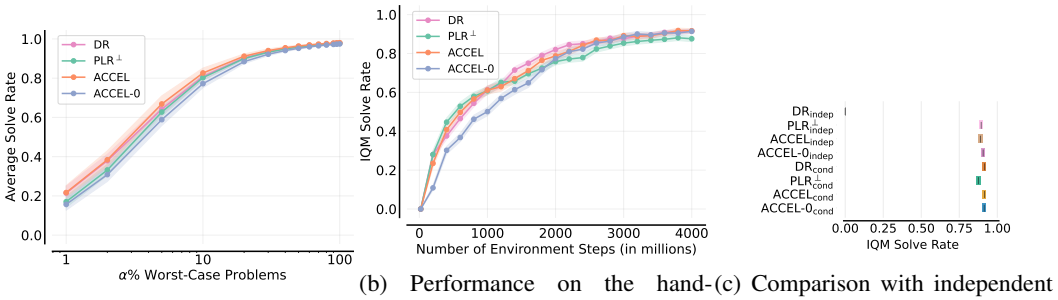
(a) 1×10^9 (b) 2×10^9 (c) 3×10^9 (d) 4×10^9

Figure 16: Samples of generated problems at different points in time (in number of environment steps) using ACCEL-0.



(a) Edit occurrence probability (ignoring non-mutated problems). (b) Average number of edits.

Figure 17: Evolution of edits in buffer problems throughout training. The frequency is weighted by the sampling probability of the associated problem.



(a) CVaR of the solve rate. (b) Performance on the hand-designed test set. (c) Comparison with independent sampling.

Figure 18: Performance of UED approaches with *level-conditioned* problem sampling.

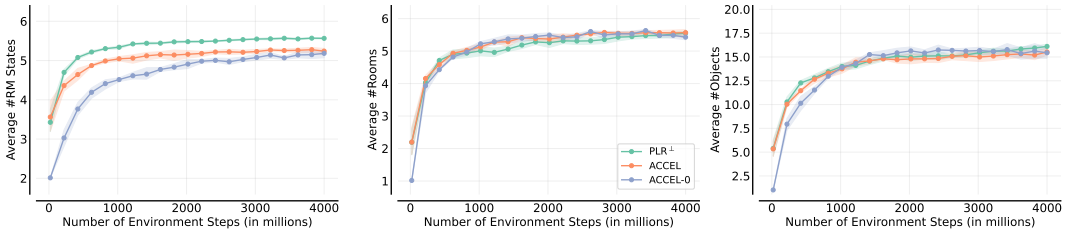


Figure 19: Emergent complexity metrics for problems in the buffer in the *level-conditioned* sampling setting.

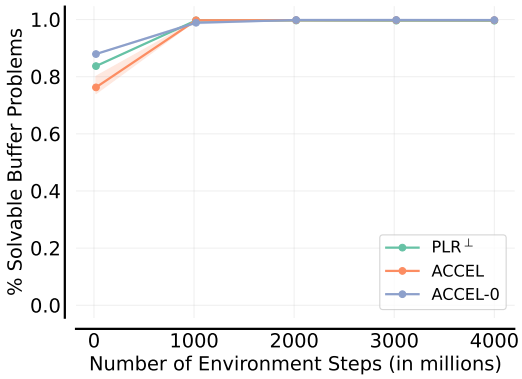
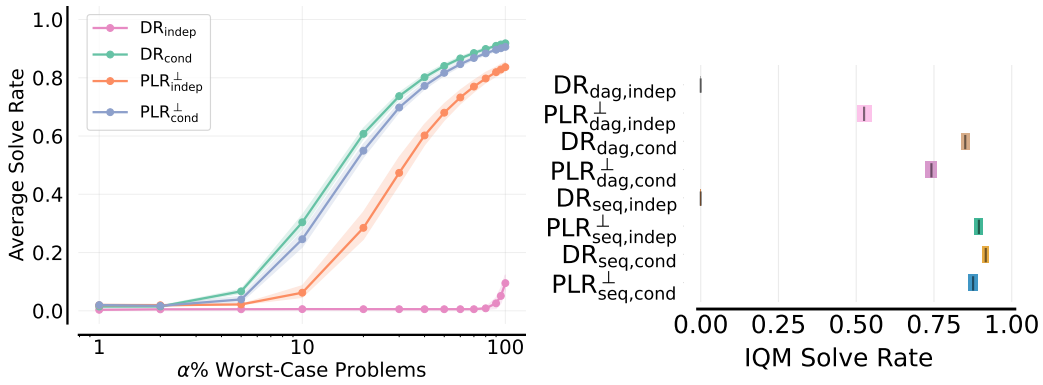


Figure 20: Fraction of solvable buffer problems throughout training in the *level-conditioned* sampling setting.

less robust than DR, albeit they are close. In line with the results on the hand-designed set obtained with sequential sampling, PLR$^{\perp}$ outperforms DR in the independent sampling setting, which is the most challenging. However, we make two key observations: (i) the performance is almost half of that obtained with sequential RMs, so increasing the training distribution complexity hinders performance, and (ii) PLR$^{\perp}$ performs slightly worse than DR in the level-conditioned setting.

Figure 22 illustrates the evolution of problems in the buffer throughout training. Since, unlike sequential RMs, DAG RMs may consist of several paths from the initial to the accepting state, we analyze whether a curriculum is induced for two new metrics: the *number of paths* and the *average path length*. Both metrics are computed via a topological sort of the RM graph. In line with the sequential setting, we observe that a curriculum is induced over all the metrics; however, some change more drastically with level-conditioned sampling (the number of states and the average path



(a) CVaR of the solve rate.

(b) Performance on the hand-designed test set.

Figure 21: Performance of UED approaches using the *random walk-based* task sampler.

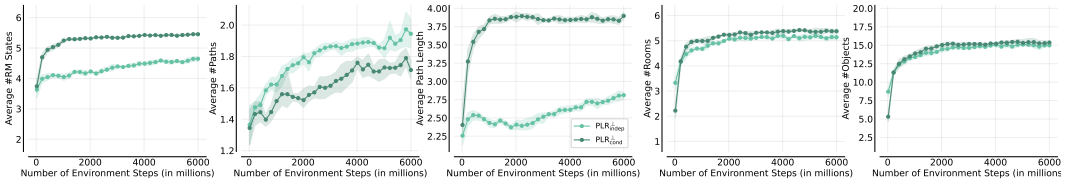
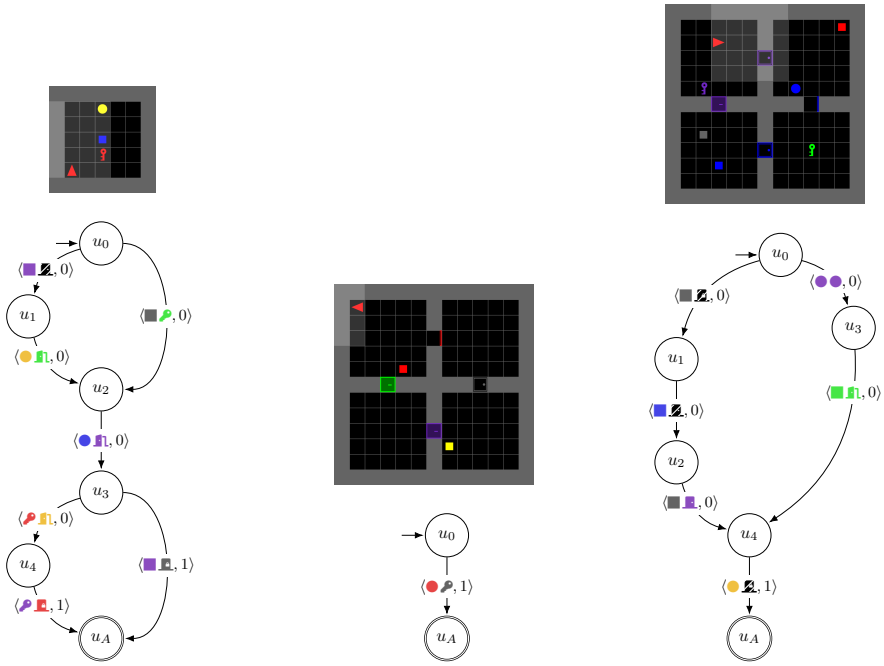


Figure 22: Emergent complexity metrics for problems in the buffer for *independent* and *level-conditioned* sampling settings using the *random-walk based* task sampler to generate DAG RMs.



(a) 2×10^9 (b) 4×10^9 (c) 6×10^9

Figure 23: Samples of DR-generated problems at different points in time (in number of environment steps) using the *random-walk based* sampler.

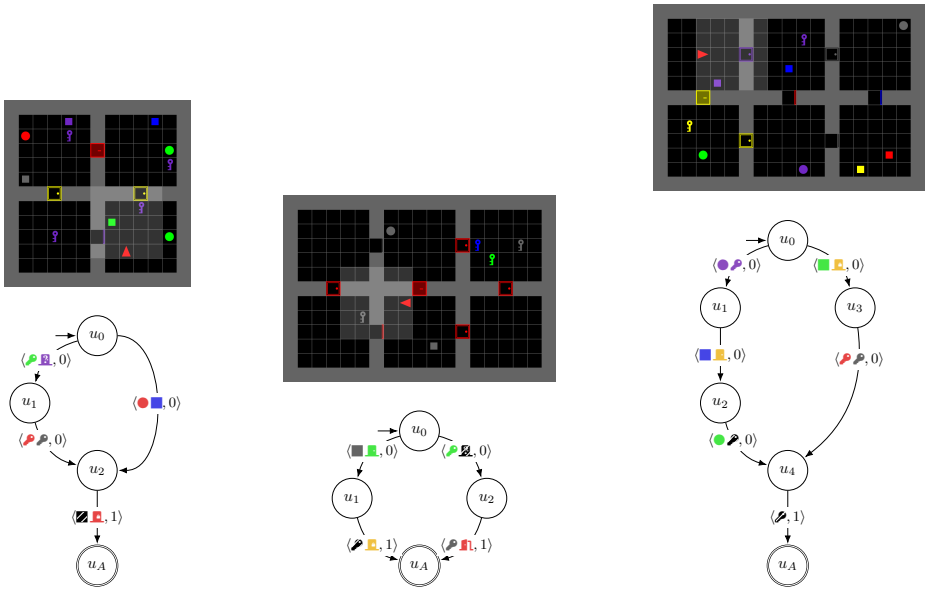
length). Figures 23 and 24 show some problems generated throughout training with DR and PLR$^{\perp}$, respectively. Negations to ensure mutual exclusivity (hence, determinism) are omitted for simplicity. While all problems are solvable for PLR$^{\perp}$, not all paths are.

Figure 25 shows the fraction of buffer problems that are solvable over time. As in the sequential RM sampling case, PLR$^{\perp}$ curates a buffer that mostly contains solvable problems.

E.7 Extended Mutation Ablation Results

We analyze how performing ablations on the *applicable edit types* and the *edit sequence length* changes the performance of ACCEL and ACCEL-0. Figure 26a shows the performance of ACCEL for different combinations of level (L), task (T), and hindsight (H) edits. By default, we perform a combination of all such edits (L+T+H). Hindsight edits are not tested in isolation since they are not always applicable, whereas at least one level/task edit is always applicable. We observe that combining level and task edits sensibly improves performance over the same edits performed in isolation. Hindsight edits enhance performance when combined with level edits alone; however, in general, performance changes induced by hindsight edits seem minimal. As previously observed in Appendix E.4, hindsight edits have little presence in the buffers.

Figure 26b illustrates the performance of ACCEL-0 with (default) and without hindsight edits. In this case, we do not ablate level and task edits since they are key to building increasingly complex



(a) 2×10^9 (b) 4×10^9 (c) 6×10^9

Figure 24: Samples of PLR$^{\perp}$-generated problems at different points in time (in number of environment steps) using the *random-walk based* sampler.

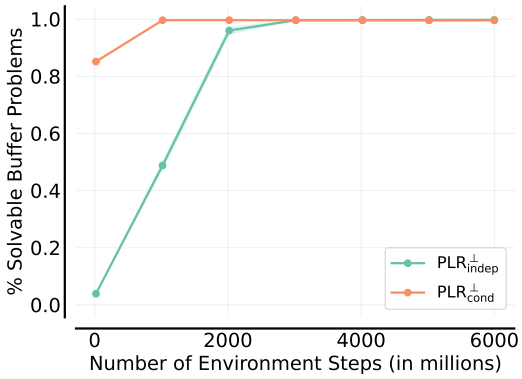


Figure 25: Fraction of solvable buffer problems throughout training using the *random walk-based* task sampler.

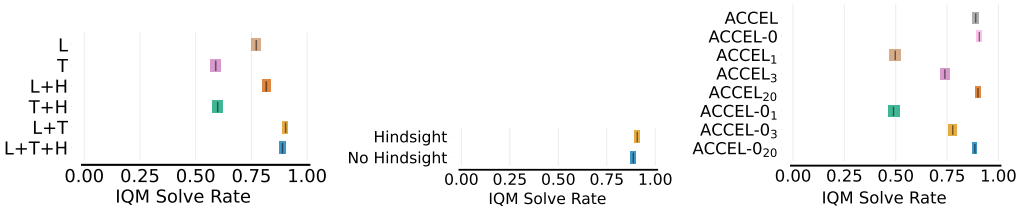
problems from the sampled ones (one room, one key, one transition). In line with the previous point, hindsight edits do not sensibly improve performance.

Figure 26c shows how performing fewer (1, 3) or more edits (20) than in our default setting changes ACCEL variants' performance. Our default setting uniformly samples between 7 and 10 edits. Decreasing the number of edits severely hinders the performance of ACCEL and ACCEL-0, especially when a single edit is performed. Increasing the number of edits to 20 has barely any effect on performance.

E.8 Task-Conditioning Ablation Results

We analyze how the behavior of PLR$^{\perp}$ changes for some ablations on task-conditioning (see Appendix C). The ablations are the following:

Vanilla Condition the policy on the index of the current RM state rather than conditioning on an RM's graph embedding.



(a) Edit type ablations in ACCEL. (b) Edit type ablations in ACCEL-0. (c) Edit sequence length ablation.

Figure 26: ACCEL and ACCEL-0 performance for ablations on the applicable edit types and length of the edit sequence.

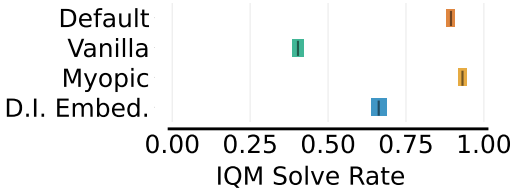


Figure 27: PLR$^{\perp}$ performance after applying different task-conditioning ablations.

Myopic Still conditions on the graph embedding, but the GCN has a single layer; hence, each RM state embedding only aggregates information from its immediate neighbors. By default, our GCN has five layers.

Domain Independent Embeddings (D.I. Embed.) Instead of exploiting domain-specific knowledge to build the literal embeddings, each literal (proposition or its negation) is embedded differently.

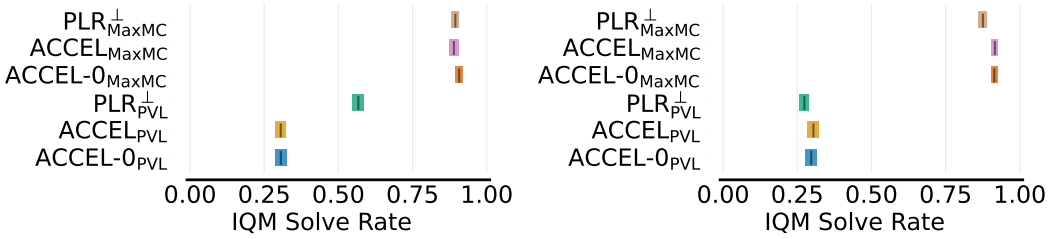
Figure 27 shows the results. We make the following observations. First, the graph embeddings provide a substantial benefit with respect to the *vanilla* embeddings since the latter cannot generalize across different tasks (the RM state index tells nothing about the task being performed). Second, the *myopic* ablation performs close to the default setting, suggesting that the latter has converged to a myopic strategy. Indeed, the performance shown for the MYOPIC problem (see Figure 4) is an additional indication: a non-myopic agent could easily solve the task 100% of the time. Third, the domain-dependent literal embeddings help improve performance with respect to the domain-agnostic ones.

E.9 Scoring Function Ablation Results

Figure 28 compares the performance of PLR$^{\perp}$, ACCEL and ACCEL-0 using two scoring functions: MaxMC (default) and the *positive value loss* [PVL; 23]. MaxMC induces a significantly better performance than PVL for both independent and level-conditioned sampling. This suggests that choosing an appropriate scoring function is key for final performance, or that PVL may require hyperparameter tuning. Performing an in-depth analysis of how scoring functions induce different training distributions is an interesting venue for future work. Along this path, Rutherford et al. [44] show that MaxMC and PVL align with success rate rather than actual regret.

E.10 Individual Problem Results

Tables 5 to 7 report the average solve rate for each of the hand-designed problems in our proposed evaluation set across different task samplers (sequential, random-walk based) and problem sampling strategies (independent, level-conditioned). The problems originally shown in Figure 4 are MYOPIC (`myopic`), PATROL (`patrol_4r_full_spec`), and CHOICE (`choice-choice_three-next_to_square_b`).



(a) Independent sampling.

(b) Level-conditioned sampling.

Figure 28: Performance of PLR$^{\perp}$, ACCEL, and ACCEL-0 using the MaxMC (default) and PVL scoring functions.

Table 5: Solve rates and 95% CIs (in brackets) for hand-designed problems with *sequential* task sampling and *independent* problem sampling.

Problem	DR	PLR$^{\perp}$	ACCEL	ACCEL-0
1r-corner_balls-back_n_forth	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.94 (0.9, 0.98)
1r-corner_balls-criss_cross	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-corner_balls-move_purple_key	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.9 (0.6, 1.0)	1.0 (1.0, 1.0)
1r-corner_balls-withholding	0.06 (0.0, 0.24)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-item_cluster-all_keys	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-item_cluster-arrangement	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.9 (0.6, 1.0)	0.96 (0.84, 1.0)
1r-item_cluster-arrangement_a	0.0 (0.0, 0.0)	0.96 (0.92, 1.0)	0.8 (0.44, 0.96)	0.98 (0.92, 1.0)
1r-item_cluster-arrangement_b	0.0 (0.0, 0.0)	0.98 (0.92, 1.0)	0.76 (0.2, 1.0)	0.92 (0.76, 1.0)
1r-key_swap-carry_yellow_blue	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-key_swap-front_red_green	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-key_swap-lookahead	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-key_swap-swap_red_green	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-arrangement	0.04 (0.0, 0.16)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-escape	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.96 (0.84, 1.0)
1r-trapped_agent-find_yellow_square	0.28 (0.08, 0.8)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-reshape_squares	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_object-big_changes	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.94 (0.76, 1.0)
1r-trapped_object-find_yellow_key	0.6 (0.2, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_object-rearrange_purple_square	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)
1r-trapped_object-sort	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-green_locked_door-ball_tribute	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-green_locked_door-bury_object	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)
2r-green_locked_door-never_unlock	0.0 (0.0, 0.0)	0.38 (0.16, 0.62)	0.3 (0.14, 0.54)	0.38 (0.1, 0.66)
2r-green_locked_door-next_square_key	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-mix	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-no_space	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-self_destructive	0.0 (0.0, 0.0)	0.98 (0.92, 1.0)	0.96 (0.84, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-the_other_side	0.0 (0.0, 0.0)	0.72 (0.34, 1.0)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)
2r-locked_blocked_0-move_blue_key	0.0 (0.0, 0.0)	0.7 (0.58, 0.82)	0.98 (0.92, 1.0)	0.82 (0.68, 0.96)
2r-locked_blocked_0-next_square_red_ball_blue	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.96 (0.92, 1.0)	1.0 (1.0, 1.0)
2r-locked_blocked_0-no_unlock	0.0 (0.0, 0.0)	0.66 (0.5, 0.8)	0.6 (0.3, 0.8)	0.66 (0.34, 0.9)
2r-locked_blocked_0-pair_squares	0.0 (0.0, 0.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-locked_blocked_1-explore	0.0 (0.0, 0.0)	0.92 (0.84, 1.0)	1.0 (1.0, 1.0)	0.9 (0.76, 1.0)
2r-locked_blocked_1-never_leave	0.0 (0.0, 0.0)	0.66 (0.38, 0.86)	0.7 (0.42, 0.96)	0.66 (0.42, 0.92)
2r-locked_blocked_1-recall	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
2r-locked_blocked_1-retrieve	0.0 (0.0, 0.0)	0.54 (0.3, 0.8)	0.92 (0.84, 0.98)	0.72 (0.4, 0.9)
2r-locked_choice-choose	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-locked_choice-pair_keys	0.0 (0.0, 0.0)	0.56 (0.34, 0.76)	0.6 (0.48, 0.78)	0.68 (0.28, 0.88)
2r-locked_choice-rearrange	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-locked_choice-toggle_door	0.0 (0.0, 0.0)	0.16 (0.04, 0.48)	0.16 (0.08, 0.24)	0.32 (0.12, 0.46)
4r-circuit-around_the_world	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
4r-circuit-pair_keys	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
4r-circuit-squares	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
4r-circuit-take_square_purple	0.0 (0.0, 0.0)	0.82 (0.68, 0.96)	0.9 (0.78, 0.98)	0.9 (0.78, 1.0)
4r-four_balls_a-adventure	0.0 (0.0, 0.0)	0.64 (0.36, 0.86)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
4r-four_balls_a-arrange_balls	0.0 (0.0, 0.0)	0.96 (0.92, 1.0)	0.36 (0.1, 0.6)	0.78 (0.6, 0.96)
4r-four_balls_a-connect_balls	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)
4r-four_balls_a-place_keys	0.0 (0.0, 0.0)	0.5 (0.36, 0.64)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)
4r-sequential_rooms_a-block_doors	0.0 (0.0, 0.0)	0.9 (0.84, 0.96)	0.7 (0.28, 0.9)	0.86 (0.76, 0.94)
4r-sequential_rooms_a-never_unlock	0.0 (0.0, 0.0)	0.26 (0.14, 0.42)	0.46 (0.26, 0.66)	0.14 (0.04, 0.4)
4r-sequential_rooms_a-pair_red_keys	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
4r-sequential_rooms_a-sort	0.0 (0.0, 0.0)	0.96 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
4r-trapped-ball_search	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.94 (0.76, 1.0)
4r-trapped-excavation	0.0 (0.0, 0.0)	0.38 (0.16, 0.64)	0.3 (0.12, 0.54)	0.12 (0.04, 0.2)
4r-trapped-key_search	0.0 (0.0, 0.0)	0.72 (0.52, 0.94)	0.98 (0.92, 1.0)	0.8 (0.44, 0.96)
4r-trapped-retrieve	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.96 (0.84, 1.0)	1.0 (1.0, 1.0)
4r-u_shape_blocked_a-all_grey	0.0 (0.0, 0.0)	0.96 (0.84, 1.0)	0.88 (0.68, 0.98)	0.92 (0.84, 0.98)
4r-u_shape_blocked_a-find_yellow_objs	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.78 (0.2, 0.98)	0.92 (0.84, 1.0)
4r-u_shape_blocked_a-pairs	0.0 (0.0, 0.0)	0.9 (0.6, 1.0)	1.0 (1.0, 1.0)	0.94 (0.76, 1.0)
4r-u_shape_blocked_a-unlock_doors	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)

Table 5: Solve rates and 95% CIs (in brackets) for hand-designed problems with *sequential* task sampling and *independent* problem sampling (continued).

Problem	DR	PLR$^{\perp}$	ACCEL	ACCEL-0
6r-corridors-arrange_keys	0.0 (0.0, 0.0)	0.72 (0.64, 0.8)	0.84 (0.74, 0.9)	0.84 (0.72, 0.96)
6r-corridors-block_doors	0.0 (0.0, 0.0)	0.78 (0.62, 0.92)	0.82 (0.62, 0.96)	0.74 (0.54, 0.88)
6r-corridors-find	0.0 (0.0, 0.0)	0.1 (0.02, 0.18)	0.1 (0.02, 0.18)	0.1 (0.0, 0.24)
6r-corridors-green_square_search	0.0 (0.0, 0.0)	0.38 (0.14, 0.56)	0.22 (0.14, 0.3)	0.1 (0.0, 0.4)
6r-key_objective-find_red_ball	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-key_objective-key_tour	0.0 (0.0, 0.0)	0.96 (0.92, 1.0)	0.96 (0.84, 1.0)	0.94 (0.84, 1.0)
6r-key_objective-pair_balls	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-key_objective-return_to_base	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-key_temple-ball_offer	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.04 (0.0, 0.16)	0.08 (0.0, 0.24)
6r-key_temple-pair_chain	0.0 (0.0, 0.0)	0.46 (0.34, 0.62)	0.52 (0.18, 0.86)	0.68 (0.4, 0.92)
6r-key_temple-ritual	0.0 (0.0, 0.0)	0.14 (0.04, 0.32)	0.38 (0.24, 0.78)	0.48 (0.2, 0.78)
6r-key_temple-unlock_chain	0.0 (0.0, 0.0)	0.52 (0.36, 0.7)	0.64 (0.54, 0.9)	0.5 (0.3, 0.84)
6r-locked_rooms-ball_sequence	0.0 (0.0, 0.0)	0.52 (0.24, 0.78)	0.24 (0.06, 0.42)	0.34 (0.2, 0.58)
6r-locked_rooms-find_blue_ball	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-locked_rooms-green_stuff	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-locked_rooms-unlock_doors	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
6r-secret_treasure-find_treasure	0.0 (0.0, 0.0)	0.28 (0.1, 0.64)	0.28 (0.04, 0.56)	0.24 (0.08, 0.56)
6r-secret_treasure-pairings	0.0 (0.0, 0.0)	0.1 (0.0, 0.32)	0.12 (0.04, 0.24)	0.32 (0.24, 0.4)
6r-secret_treasure-take_the_key	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-secret_treasure-unlock_all	0.0 (0.0, 0.0)	0.84 (0.68, 0.94)	0.54 (0.32, 0.78)	0.62 (0.34, 0.86)
anyorder_2o	0.0 (0.0, 0.0)	0.98 (0.92, 1.0)	0.94 (0.84, 1.0)	0.96 (0.92, 1.0)
anyorder_3o	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_four-carrying_keys	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_four-front_green_square	0.0 (0.0, 0.0)	0.46 (0.3, 0.6)	0.46 (0.28, 0.86)	0.48 (0.22, 0.74)
choice-choice_four-front_red_square	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.96 (0.84, 1.0)	0.98 (0.92, 1.0)
choice-choice_four-no_unlocking	0.0 (0.0, 0.0)	0.36 (0.14, 0.5)	0.56 (0.26, 0.86)	0.62 (0.52, 0.72)
choice-choice_four-pair_blue_red_objs	0.0 (0.0, 0.0)	0.14 (0.06, 0.24)	0.32 (0.12, 0.42)	0.4 (0.2, 0.56)
choice-choice_four-patience_test	0.0 (0.0, 0.0)	0.92 (0.76, 1.0)	0.96 (0.92, 1.0)	0.9 (0.82, 0.98)
choice-choice_four-red_ball_square	0.0 (0.0, 0.0)	0.26 (0.16, 0.34)	0.14 (0.02, 0.48)	0.14 (0.1, 0.26)
choice-choice_four-two_red_squares	0.0 (0.0, 0.0)	0.58 (0.44, 0.72)	0.58 (0.36, 0.84)	0.58 (0.42, 0.78)
choice-choice_one-arrange_squares	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-key_to_square	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-move_everything	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-retrieve_square	0.0 (0.0, 0.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-toggle_door_squares	0.0 (0.0, 0.0)	0.8 (0.2, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-unlock_door	0.18 (0.04, 0.38)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-unlock_door_specific	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-visit_all_squares	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_three-front_ball_blue	0.0 (0.0, 0.0)	0.48 (0.22, 0.72)	0.36 (0.14, 0.72)	0.3 (0.12, 0.44)
choice-choice_three-front_ball_purple	0.0 (0.0, 0.0)	0.5 (0.24, 0.76)	0.56 (0.42, 0.7)	0.56 (0.36, 0.78)
choice-choice_three-next_to_square	0.0 (0.0, 0.0)	0.98 (0.92, 1.0)	0.8 (0.52, 0.96)	0.9 (0.82, 0.98)
choice-choice_three-next_to_square_a	0.0 (0.0, 0.0)	0.96 (0.92, 1.0)	0.82 (0.68, 0.96)	0.8 (0.72, 0.94)
choice-choice_three-next_to_square_b	0.0 (0.0, 0.0)	0.58 (0.38, 0.8)	0.76 (0.66, 0.92)	0.68 (0.44, 0.86)
choice-choice_three-next_to_squares	0.0 (0.0, 0.0)	0.94 (0.76, 1.0)	0.8 (0.68, 0.92)	0.86 (0.6, 0.96)
choice-choice_three-next_to_squares_a	0.0 (0.0, 0.0)	0.82 (0.68, 0.92)	0.78 (0.6, 0.92)	0.92 (0.76, 1.0)
choice-choice_three-unlocking_sequence	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.96 (0.84, 1.0)	1.0 (1.0, 1.0)
choice-choice_two-avoid_unlock	0.0 (0.0, 0.0)	0.04 (0.0, 0.08)	0.08 (0.02, 0.16)	0.02 (0.0, 0.08)
choice-choice_two-avoid_unlock_hard	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)
choice-choice_two-avoid_unlock_impossible	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)
choice-choice_two-front_square_green	0.12 (0.0, 0.48)	0.82 (0.58, 1.0)	0.88 (0.68, 0.96)	0.86 (0.6, 1.0)
choice-choice_two-front_squares	0.0 (0.0, 0.0)	0.34 (0.1, 0.64)	0.12 (0.02, 0.24)	0.14 (0.0, 0.4)
choice-choice_two-front_squares_hard	0.0 (0.0, 0.0)	0.3 (0.16, 0.48)	0.34 (0.14, 0.54)	0.32 (0.1, 0.62)
choice-choice_two-intermediate_door_unlock	0.0 (0.0, 0.0)	0.72 (0.52, 0.88)	0.56 (0.22, 0.78)	0.88 (0.76, 0.98)
choice-choice_two-next_key_square	0.0 (0.0, 0.0)	0.08 (0.02, 0.16)	0.28 (0.04, 0.56)	0.26 (0.06, 0.54)
choice-choice_two-next_to_squares_0	0.0 (0.0, 0.0)	0.5 (0.2, 0.78)	0.24 (0.06, 0.42)	0.36 (0.2, 0.52)
choice-choice_two-next_to_squares_1	0.0 (0.0, 0.0)	0.36 (0.12, 0.52)	0.34 (0.18, 0.46)	0.26 (0.1, 0.54)
myopic	0.0 (0.0, 0.0)	0.4 (0.3, 0.62)	0.28 (0.18, 0.52)	0.48 (0.22, 0.74)
patrol_4r_full_spec	0.0 (0.0, 0.0)	0.82 (0.36, 0.98)	0.56 (0.2, 0.84)	0.82 (0.6, 1.0)
patrol_4r_overspec	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)
patrol_4r_underspec	0.0 (0.0, 0.0)	0.6 (0.22, 0.86)	0.04 (0.0, 0.16)	0.18 (0.04, 0.48)
patrol_4r_underspec-1	0.0 (0.0, 0.0)	0.58 (0.24, 0.82)	0.24 (0.12, 0.44)	0.32 (0.12, 0.62)
patrol_6r	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
patrol_6r_objects	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
patrol_6r_objects_obstacles	0.0 (0.0, 0.0)	0.9 (0.84, 0.95)	0.88 (0.82, 0.94)	0.9 (0.76, 1.0)
patrol_6r_obstacles	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
putnext_1r	0.0 (0.0, 0.0)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)
putnext_2r-0	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
putnext_2r-1	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
putnext_2r_distractor-0	0.0 (0.0, 0.0)	0.68 (0.34, 0.86)	0.66 (0.36, 0.88)	0.86 (0.52, 1.0)
putnext_2r_distractor-1	0.0 (0.0, 0.0)	0.42 (0.2, 0.6)	0.56 (0.26, 0.86)	0.78 (0.66, 0.88)
putnext_4r	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
unlocktounlock_overspec	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
unlocktounlock_underspec	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)
walls-chunks-blue_ball	0.34 (0.12, 0.58)	0.82 (0.44, 0.96)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)
walls-chunks-green_ball	0.04 (0.0, 0.16)	0.96 (0.92, 1.0)	0.7 (0.52, 0.88)	0.92 (0.84, 0.98)
walls-chunks-grey_ball	0.42 (0.18, 0.72)	0.96 (0.92, 1.0)	0.98 (0.92, 1.0)	0.96 (0.84, 1.0)
walls-chunks-purple_ball	0.0 (0.0, 0.0)	0.88 (0.76, 0.96)	0.8 (0.68, 0.9)	0.72 (0.52, 0.9)
walls-chunks-red_ball	0.04 (0.0, 0.16)	0.12 (0.0, 0.4)	0.04 (0.0, 0.16)	0.16 (0.0, 0.48)
walls-chunks-yellow_ball	0.0 (0.0, 0.0)	0.52 (0.2, 0.74)	0.42 (0.1, 0.8)	0.56 (0.28, 0.78)
walls-maze_4r-red_ball	0.0 (0.0, 0.0)	0.04 (0.0, 0.16)	0.04 (0.0, 0.08)	0.02 (0.0, 0.08)
walls-maze_4r-red_key	0.0 (0.0, 0.0)	0.48 (0.1, 0.84)	0.52 (0.26, 0.78)	0.26 (0.06, 0.82)
walls-maze_4r-red_square	0.0 (0.0, 0.0)	0.08 (0.0, 0.24)	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)
walls-maze_6r-green_ball	0.0 (0.0, 0.0)	0.08 (0.0, 0.24)	0.36 (0.08, 0.68)	0.06 (0.0, 0.14)
walls-maze_6r-green_key	0.0 (0.0, 0.0)	0.06 (0.0, 0.24)	0.02 (0.0, 0.08)	0.08 (0.0, 0.32)

Table 5: Solve rates and 95% CIs (in brackets) for hand-designed problems with *sequential* task sampling and *independent* problem sampling (continued).

Problem	DR	PLR$^{\perp}$	ACCEL	ACCEL-0
walls-maze_6r-green_square	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.06 (0.02, 0.1)	0.04 (0.0, 0.16)
walls-maze_9r-blue_ball	0.0 (0.0, 0.0)	0.28 (0.04, 0.56)	0.22 (0.08, 0.48)	0.3 (0.0, 0.66)
walls-maze_9r-blue_key	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)
walls-maze_9r-blue_square	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)
walls-spiral_4r-purple_key	0.02 (0.0, 0.08)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)	0.04 (0.0, 0.08)
walls-spiral_9r-grey_key	0.18 (0.0, 0.72)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)

Table 6: Solve rates and 95% CIs (in brackets) for hand-designed problems with *sequential* task sampling and *level-conditioned* problem sampling.

Problem	DR	PLR$^{\perp}$	ACCEL	ACCEL-0
1r-corner_balls-back_n_forth	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-corner_balls-criss_cross	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-corner_balls-move_purple_key	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-corner_balls-withholding	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-item_cluster-all_keys	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-item_cluster-arrangement	1.0 (1.0, 1.0)	0.88 (0.52, 1.0)	1.0 (1.0, 1.0)	0.96 (0.84, 1.0)
1r-item_cluster-arrangement_a	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.94 (0.84, 1.0)
1r-item_cluster-arrangement_b	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	0.96 (0.92, 1.0)
1r-key_swap-carry_yellow_blue	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-key_swap-front_red_green	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-key_swap-lookahead	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-key_swap-swap_red_green	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-arrangement	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-escape	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-find_yellow_square	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-reshape_squares	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_object-big_changes	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
1r-trapped_object-find_yellow_key	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_object-rearrange_purple_square	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
1r-trapped_object-sort	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-green_locked_door-ball_tribute	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-green_locked_door-bury_object	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-green_locked_door-never_unlock	0.58 (0.34, 0.78)	0.82 (0.68, 0.94)	0.76 (0.5, 0.9)	0.48 (0.26, 0.68)
2r-green_locked_door-next_square_key	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-mix	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-no_space	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-self_destructive	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-the_other_side	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.88 (0.52, 1.0)	0.94 (0.76, 1.0)
2r-locked_blocked_0-move_blue_key	0.86 (0.8, 0.94)	0.88 (0.6, 1.0)	0.96 (0.92, 1.0)	0.92 (0.84, 1.0)
2r-locked_blocked_0-next_square_red_ball_blue	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)	0.96 (0.84, 1.0)
2r-locked_blocked_0-no_unlock	0.64 (0.54, 0.74)	0.44 (0.24, 0.58)	0.88 (0.68, 0.96)	0.82 (0.76, 0.88)
2r-locked_blocked_0-pair_squares	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-locked_blocked_1-explore	0.92 (0.68, 1.0)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)	0.92 (0.68, 1.0)
2r-locked_blocked_1-never_leave	0.48 (0.3, 0.74)	0.7 (0.4, 0.94)	0.88 (0.82, 0.9)	0.66 (0.44, 0.86)
2r-locked_blocked_1-recall	0.9 (0.84, 0.96)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-locked_blocked_1-retrieve	0.64 (0.5, 0.78)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)	0.96 (0.84, 1.0)
2r-locked_choice-choose	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-locked_choice-pair_keys	0.84 (0.8, 0.88)	0.56 (0.4, 0.7)	0.68 (0.32, 0.82)	0.82 (0.68, 0.96)
2r-locked_choice-rearrange	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-locked_choice-toggle_door	0.18 (0.06, 0.32)	0.3 (0.18, 0.38)	0.62 (0.28, 0.9)	0.36 (0.2, 0.52)
4r-circuit-around_the_world	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
4r-circuit-pair_keys	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
4r-circuit-squares	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
4r-circuit-take_square_purple	0.94 (0.84, 1.0)	0.8 (0.56, 0.96)	0.92 (0.84, 0.98)	0.84 (0.76, 0.94)
4r-four_balls_a-adventure	0.96 (0.92, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
4r-four_balls_a-arrange_balls	0.72 (0.53, 0.92)	0.48 (0.32, 0.72)	0.36 (0.12, 0.6)	0.64 (0.36, 0.8)
4r-four_balls_a-connect_balls	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	0.96 (0.92, 1.0)	1.0 (1.0, 1.0)
4r-four_balls_a-place_keys	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	0.96 (0.92, 1.0)
4r-sequential_rooms_a-block_doors	0.72 (0.52, 0.82)	0.7 (0.2, 0.94)	0.78 (0.72, 0.86)	0.74 (0.58, 0.86)
4r-sequential_rooms_a-never_unlock	0.28 (0.1, 0.42)	0.2 (0.04, 0.56)	0.2 (0.04, 0.66)	0.02 (0.0, 0.08)
4r-sequential_rooms_a-pair_red_keys	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
4r-sequential_rooms_a-sort	0.88 (0.6, 1.0)	0.92 (0.76, 1.0)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)
4r-trapped-ball_search	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.96 (0.84, 1.0)
4r-trapped-excavation	0.58 (0.36, 0.78)	0.14 (0.06, 0.24)	0.06 (0.02, 0.1)	0.38 (0.2, 0.64)
4r-trapped-key_search	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)	0.96 (0.92, 1.0)	0.9 (0.6, 1.0)
4r-trapped-retrieve	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)
4r-u_shape_blocked_a-all_grey	1.0 (1.0, 1.0)	0.96 (0.84, 1.0)	0.9 (0.78, 0.96)	0.96 (0.84, 1.0)
4r-u_shape_blocked_a-find_yellow_objs	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	0.92 (0.68, 1.0)	1.0 (1.0, 1.0)
4r-u_shape_blocked_a-pairs	1.0 (1.0, 1.0)	0.9 (0.6, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
4r-u_shape_blocked_a-unlock_doors	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-corridors-arrange_keys	0.66 (0.26, 0.84)	0.9 (0.84, 0.96)	0.9 (0.78, 1.0)	0.66 (0.44, 0.86)
6r-corridors-block_doors	0.78 (0.68, 0.92)	0.78 (0.62, 0.94)	0.84 (0.6, 0.96)	0.84 (0.6, 0.96)
6r-corridors-find	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)	0.06 (0.0, 0.24)	0.04 (0.0, 0.16)
6r-corridors-green_square_search	0.1 (0.02, 0.22)	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.12 (0.0, 0.24)
6r-key_objective-find_red_ball	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-key_objective-key_tour	0.94 (0.84, 1.0)	0.68 (0.5, 0.82)	0.9 (0.78, 1.0)	0.78 (0.64, 0.92)

Table 6: Solve rates and 95% CIs (in brackets) for hand-designed problems with *sequential* task sampling and *level-conditioned* problem sampling.

Problem	DR	PLR$^{\perp}$	ACCEL	ACCEL-0
6r-key_objective-pair_balls	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-key_objective-return_to_base	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
6r-key_temple-ball_offer	0.02 (0.0, 0.08)	0.04 (0.0, 0.08)	0.04 (0.0, 0.08)	0.04 (0.0, 0.08)
6r-key_temple-pair_chain	0.74 (0.62, 0.86)	0.58 (0.32, 0.84)	0.5 (0.21, 0.62)	0.62 (0.46, 0.8)
6r-key_temple-ritual	0.64 (0.44, 0.86)	0.56 (0.4, 0.84)	0.6 (0.22, 0.9)	0.48 (0.2, 0.6)
6r-key_temple-unlock_chain	0.52 (0.34, 0.7)	0.16 (0.06, 0.32)	0.46 (0.26, 0.74)	0.38 (0.2, 0.64)
6r-locked_rooms-ball_sequence	0.78 (0.58, 0.88)	0.3 (0.04, 0.66)	0.34 (0.06, 0.62)	0.2 (0.04, 0.48)
6r-locked_rooms-find_blue_ball	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-locked_rooms-green_stuff	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-locked_rooms-unlock_doors	0.94 (0.76, 1.0)	0.86 (0.74, 0.98)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
6r-secret_treasure-find_treasure	0.1 (0.02, 0.18)	0.06 (0.0, 0.14)	0.14 (0.02, 0.48)	0.1 (0.04, 0.16)
6r-secret_treasure-pairings	0.1 (0.02, 0.18)	0.24 (0.14, 0.4)	0.5 (0.16, 0.7)	0.16 (0.0, 0.48)
6r-secret_treasure-take_the_key	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-secret_treasure-unlock_all	0.4 (0.2, 0.62)	0.34 (0.18, 0.76)	0.42 (0.16, 0.64)	0.44 (0.2, 0.74)
anyorder_2o	0.76 (0.48, 1.0)	0.9 (0.6, 1.0)	0.94 (0.84, 1.0)	0.96 (0.92, 1.0)
anyorder_3o	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.94 (0.84, 1.0)
choice-choice_four-carrying_keys	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_four-front_green_square	0.56 (0.46, 0.72)	0.56 (0.4, 0.78)	0.52 (0.18, 0.84)	0.5 (0.22, 0.76)
choice-choice_four-front_red_square	1.0 (1.0, 1.0)	0.9 (0.78, 0.98)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)
choice-choice_four-no_unlocking	0.62 (0.18, 0.8)	0.6 (0.48, 0.68)	0.7 (0.54, 0.86)	0.58 (0.24, 0.92)
choice-choice_four-pair_blue_red_objs	0.14 (0.04, 0.4)	0.2 (0.08, 0.48)	0.28 (0.18, 0.5)	0.24 (0.14, 0.36)
choice-choice_four-patience_test	0.7 (0.58, 0.8)	0.8 (0.68, 0.92)	0.86 (0.72, 0.94)	0.92 (0.84, 0.98)
choice-choice_four-red_ball_square	0.08 (0.0, 0.24)	0.12 (0.04, 0.24)	0.12 (0.04, 0.24)	0.22 (0.2, 0.28)
choice-choice_four-two_red_squares	0.7 (0.52, 0.88)	0.5 (0.3, 0.68)	0.56 (0.46, 0.7)	0.56 (0.26, 0.8)
choice-choice_one-arrange_squares	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-key_to_square	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-move_everything	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-retrieve_square	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-toggle_door_squares	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-unlock_door	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-unlock_door_specific	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-visit_all_squares	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_three-front_ball_blue	0.48 (0.4, 0.64)	0.34 (0.2, 0.56)	0.48 (0.3, 0.68)	0.46 (0.28, 0.64)
choice-choice_three-front_ball_purple	0.52 (0.38, 0.66)	0.3 (0.18, 0.6)	0.54 (0.42, 0.66)	0.5 (0.36, 0.68)
choice-choice_three-next_to_square	0.94 (0.84, 1.0)	0.84 (0.68, 0.92)	0.9 (0.78, 0.98)	0.94 (0.84, 1.0)
choice-choice_three-next_to_square_a	0.7 (0.46, 0.86)	0.8 (0.68, 0.92)	0.7 (0.44, 0.88)	0.84 (0.6, 0.96)
choice-choice_three-next_to_square_b	0.58 (0.42, 0.7)	0.66 (0.56, 0.76)	0.72 (0.64, 0.84)	0.64 (0.52, 0.8)
choice-choice_three-next_to_squares	0.92 (0.76, 1.0)	0.72 (0.56, 0.8)	0.82 (0.7, 0.94)	0.92 (0.84, 0.98)
choice-choice_three-next_to_squares_a	0.88 (0.82, 0.94)	0.8 (0.66, 0.94)	0.9 (0.82, 0.98)	0.9 (0.68, 1.0)
choice-choice_three-unlocking_sequence	0.96 (0.92, 1.0)	0.88 (0.76, 0.96)	0.98 (0.92, 1.0)	0.98 (0.92, 1.0)
choice-choice_two-avoid_unlock	0.12 (0.04, 0.18)	0.08 (0.02, 0.14)	0.04 (0.0, 0.08)	0.12 (0.04, 0.18)
choice-choice_two-avoid_unlock_hard	0.04 (0.0, 0.16)	0.06 (0.02, 0.1)	0.02 (0.0, 0.08)	0.02 (0.0, 0.08)
choice-choice_two-avoid_unlock_impossible	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)	0.04 (0.0, 0.16)
choice-choice_two-front_square_green	1.0 (1.0, 1.0)	0.88 (0.68, 1.0)	0.9 (0.6, 1.0)	0.96 (0.84, 1.0)
choice-choice_two-front_squares	0.78 (0.68, 0.92)	0.24 (0.04, 0.52)	0.24 (0.08, 0.66)	0.38 (0.12, 0.64)
choice-choice_two-front_squares_hard	0.82 (0.72, 0.92)	0.3 (0.1, 0.56)	0.32 (0.14, 0.5)	0.44 (0.16, 0.8)
choice-choice_two-intermediate_door_unlock	0.94 (0.84, 1.0)	0.8 (0.36, 1.0)	0.74 (0.36, 0.96)	0.68 (0.46, 0.8)
choice-choice_two-next_key_square	0.5 (0.28, 0.72)	0.22 (0.1, 0.4)	0.26 (0.12, 0.4)	0.6 (0.42, 0.78)
choice-choice_two-next_to_squares_0	0.82 (0.74, 0.88)	0.22 (0.14, 0.34)	0.34 (0.14, 0.54)	0.62 (0.46, 0.86)
choice-choice_two-next_to_squares_1	0.58 (0.44, 0.72)	0.44 (0.3, 0.54)	0.3 (0.06, 0.56)	0.58 (0.38, 0.68)
myopic	0.34 (0.16, 0.64)	0.52 (0.24, 0.82)	0.52 (0.32, 0.82)	0.42 (0.28, 0.58)
patrol_4r_full_spec	0.98 (0.92, 1.0)	0.5 (0.16, 0.86)	0.84 (0.44, 0.98)	0.88 (0.76, 0.96)
patrol_4r_overspec	1.0 (1.0, 1.0)	0.86 (0.52, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)
patrol_4r_underspec	0.94 (0.84, 1.0)	0.2 (0.04, 0.48)	0.42 (0.24, 0.64)	0.2 (0.08, 0.32)
patrol_4r_underspec-1	0.96 (0.92, 1.0)	0.3 (0.12, 0.54)	0.42 (0.18, 0.7)	0.28 (0.12, 0.42)
patrol_6r	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
patrol_6r_objects	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)
patrol_6r_objects_obstacles	0.98 (0.92, 1.0)	0.94 (0.84, 1.0)	0.62 (0.4, 0.84)	0.96 (0.92, 1.0)
patrol_6r_obstacles	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
putnext_1r	1.0 (1.0, 1.0)	0.92 (0.76, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
putnext_2r-0	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
putnext_2r-1	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
putnext_2r_distractor-0	0.76 (0.72, 0.8)	0.86 (0.76, 0.94)	0.76 (0.66, 0.84)	0.88 (0.76, 0.96)
putnext_2r_distractor-1	0.62 (0.36, 0.8)	0.8 (0.42, 0.94)	0.46 (0.26, 0.54)	0.66 (0.52, 0.8)
putnext_4r	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
unlocktounlock_overspec	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
unlocktounlock_underspec	0.96 (0.92, 1.0)	0.84 (0.6, 0.96)	0.92 (0.68, 1.0)	0.86 (0.6, 1.0)
walls-chunks-blue_ball	0.7 (0.36, 0.88)	0.92 (0.76, 1.0)	0.86 (0.6, 0.96)	1.0 (1.0, 1.0)
walls-chunks-green_ball	0.22 (0.16, 0.28)	0.8 (0.68, 0.92)	0.8 (0.62, 0.94)	0.94 (0.84, 1.0)
walls-chunks-grey_ball	0.92 (0.76, 1.0)	0.96 (0.84, 1.0)	0.96 (0.92, 1.0)	0.98 (0.92, 1.0)
walls-chunks-purple_ball	0.3 (0.2, 0.42)	0.56 (0.3, 0.7)	0.82 (0.76, 0.88)	0.84 (0.6, 0.96)
walls-chunks-red_ball	0.04 (0.0, 0.08)	0.08 (0.02, 0.14)	0.2 (0.04, 0.64)	0.14 (0.04, 0.4)
walls-chunks-yellow_ball	0.26 (0.16, 0.36)	0.2 (0.06, 0.38)	0.54 (0.26, 0.78)	0.34 (0.16, 0.52)
walls-maze_4r-red_ball	0.02 (0.0, 0.08)	0.06 (0.0, 0.24)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)
walls-maze_4r-red_key	0.2 (0.08, 0.48)	0.36 (0.16, 0.48)	0.5 (0.18, 0.86)	0.56 (0.28, 0.76)
walls-maze_4r-red_square	0.04 (0.0, 0.16)	0.16 (0.04, 0.48)	0.1 (0.0, 0.32)	0.28 (0.08, 0.82)
walls-maze_6r-green_ball	0.0 (0.0, 0.0)	0.34 (0.12, 0.68)	0.22 (0.0, 0.72)	0.34 (0.04, 0.66)
walls-maze_6r-green_key	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.08 (0.0, 0.24)	0.0 (0.0, 0.0)
walls-maze_6r-green_square	0.0 (0.0, 0.0)	0.06 (0.0, 0.14)	0.02 (0.0, 0.08)	0.1 (0.0, 0.32)
walls-maze_9r-blue_ball	0.0 (0.0, 0.0)	0.24 (0.08, 0.4)	0.06 (0.0, 0.24)	0.5 (0.24, 0.74)
walls-maze_9r-blue_key	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.14 (0.0, 0.4)
walls-maze_9r-blue_square	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)
walls-spiral_4r-purple_key	0.0 (0.0, 0.0)	0.12 (0.02, 0.4)	0.0 (0.0, 0.0)	0.1 (0.0, 0.32)
walls-spiral_9r-grey_key	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)

Table 7: Solve rates and 95% CIs (in brackets) for hand-designed problems with *random walk-based* task sampling.

Problem	DR_{indep}	$PLR^{\perp}_{indep}$	DR_{cond}	$PLR^{\perp}_{cond}$
1r-corner_balls-back_n_forth	0.0 (0.0, 0.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-corner_balls-criss_cross	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-corner_balls-move_purple_key	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-corner_balls-withholding	0.16 (0.0, 0.36)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-item_cluster-all_keys	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-item_cluster-arrangement	0.12 (0.0, 0.48)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-item_cluster-arrangement_a	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-item_cluster-arrangement_b	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-key_swap-carry_yellow_blue	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-key_swap-front_red_green	0.3 (0.08, 0.72)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-key_swap-lookahead	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-key_swap-swap_red_green	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-arrangement	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-escape	0.0 (0.0, 0.0)	0.9 (0.6, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-find_yellow_square	0.02 (0.0, 0.08)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_agent-reshape_squares	0.0 (0.0, 0.0)	0.92 (0.76, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_object-big_changes	0.08 (0.0, 0.32)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_object-find_yellow_key	0.4 (0.08, 0.8)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_object-rearrange_purple_square	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
1r-trapped_object-sort	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-green_locked_door-ball_tribute	0.0 (0.0, 0.0)	0.18 (0.0, 0.56)	1.0 (1.0, 1.0)	0.82 (0.66, 0.98)
2r-green_locked_door-bury_object	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-green_locked_door-never_unlock	0.0 (0.0, 0.0)	0.92 (0.84, 0.98)	0.64 (0.36, 0.88)	0.74 (0.66, 0.92)
2r-green_locked_door-next_square_key	0.0 (0.0, 0.0)	0.78 (0.54, 0.94)	0.98 (0.92, 1.0)	0.96 (0.84, 1.0)
2r-key_barrier-mix	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-no_space	0.0 (0.0, 0.0)	0.82 (0.44, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-self_destructive	0.0 (0.0, 0.0)	0.96 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-key_barrier-the_other_side	0.0 (0.0, 0.0)	0.74 (0.64, 0.84)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
2r-locked_blocked_0-move_blue_key	0.0 (0.0, 0.0)	0.2 (0.08, 0.44)	0.8 (0.58, 0.9)	0.72 (0.6, 0.88)
2r-locked_blocked_0-next_square_red_ball_blue	0.0 (0.0, 0.0)	0.2 (0.0, 0.44)	0.94 (0.9, 0.98)	0.76 (0.58, 0.94)
2r-locked_blocked_0-no_unlock	0.0 (0.0, 0.0)	0.76 (0.46, 0.92)	0.34 (0.2, 0.48)	0.52 (0.26, 0.74)
2r-locked_blocked_0-pair_squares	0.0 (0.0, 0.0)	0.52 (0.46, 0.62)	0.96 (0.84, 1.0)	1.0 (1.0, 1.0)
2r-locked_blocked_1-explore	0.0 (0.0, 0.0)	0.76 (0.42, 0.88)	0.84 (0.44, 1.0)	0.9 (0.6, 1.0)
2r-locked_blocked_1-never_leave	0.0 (0.0, 0.0)	0.88 (0.76, 0.96)	0.56 (0.3, 0.84)	0.8 (0.62, 0.94)
2r-locked_blocked_1-recall	0.0 (0.0, 0.0)	0.3 (0.1, 0.56)	0.86 (0.6, 0.96)	0.92 (0.76, 1.0)
2r-locked_blocked_1-retrieve	0.0 (0.0, 0.0)	0.2 (0.0, 0.44)	0.66 (0.52, 0.8)	0.84 (0.52, 0.96)
2r-locked_choice-choose	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	0.92 (0.76, 1.0)	0.98 (0.92, 1.0)
2r-locked_choice-pair_keys	0.0 (0.0, 0.0)	0.24 (0.08, 0.36)	0.84 (0.68, 0.96)	0.44 (0.16, 0.6)
2r-locked_choice-rearrange	0.0 (0.0, 0.0)	0.76 (0.44, 0.9)	0.9 (0.78, 0.98)	0.96 (0.92, 1.0)
2r-locked_choice-toggle_door	0.0 (0.0, 0.0)	0.06 (0.0, 0.16)	0.24 (0.1, 0.4)	0.32 (0.16, 0.5)
4r-circuit-around_the_world	0.0 (0.0, 0.0)	0.58 (0.32, 0.84)	1.0 (1.0, 1.0)	0.94 (0.9, 0.98)
4r-circuit-pair_keys	0.0 (0.0, 0.0)	0.54 (0.24, 0.8)	0.98 (0.92, 1.0)	0.82 (0.58, 1.0)
4r-circuit-squares	0.0 (0.0, 0.0)	0.86 (0.6, 0.96)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
4r-circuit-take_square_purple	0.0 (0.0, 0.0)	0.92 (0.76, 1.0)	1.0 (1.0, 1.0)	0.96 (0.92, 1.0)
4r-four_balls_a-adventure	0.0 (0.0, 0.0)	0.86 (0.76, 0.94)	0.96 (0.92, 1.0)	1.0 (1.0, 1.0)
4r-four_balls_a-arrange_balls	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.4 (0.2, 0.62)	0.04 (0.0, 0.08)
4r-four_balls_a-connect_balls	0.0 (0.0, 0.0)	0.34 (0.04, 0.66)	0.96 (0.84, 1.0)	0.86 (0.76, 0.94)
4r-four_balls_a-place_keys	0.0 (0.0, 0.0)	0.4 (0.14, 0.78)	0.9 (0.68, 1.0)	0.88 (0.72, 1.0)
4r-sequential_rooms_a-block_doors	0.0 (0.0, 0.0)	0.58 (0.22, 0.82)	0.6 (0.34, 0.86)	0.4 (0.14, 0.66)
4r-sequential_rooms_a-never_unlock	0.0 (0.0, 0.0)	0.06 (0.0, 0.14)	0.32 (0.14, 0.54)	0.36 (0.08, 0.72)
4r-sequential_rooms_a-pair_red_keys	0.0 (0.0, 0.0)	0.52 (0.38, 0.88)	1.0 (1.0, 1.0)	0.88 (0.76, 1.0)
4r-sequential_rooms_a-sort	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.62 (0.54, 0.68)	0.36 (0.12, 0.58)
4r-trapped-ball_search	0.0 (0.0, 0.0)	0.68 (0.4, 0.8)	0.92 (0.84, 0.98)	0.9 (0.84, 0.96)
4r-trapped-excavation	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.32 (0.12, 0.62)	0.16 (0.04, 0.32)
4r-trapped-key_search	0.0 (0.0, 0.0)	0.4 (0.24, 0.68)	0.98 (0.92, 1.0)	0.76 (0.44, 0.9)
4r-trapped-retrieve	0.0 (0.0, 0.0)	0.92 (0.84, 0.98)	1.0 (1.0, 1.0)	0.92 (0.68, 1.0)
4r-u_shape_blocked_a-all_grey	0.0 (0.0, 0.0)	0.08 (0.0, 0.32)	0.94 (0.84, 1.0)	0.42 (0.22, 0.56)
4r-u_shape_blocked_a-find_yellow_objs	0.0 (0.0, 0.0)	0.16 (0.04, 0.32)	0.96 (0.92, 1.0)	0.9 (0.68, 1.0)
4r-u_shape_blocked_a-pairs	0.0 (0.0, 0.0)	0.34 (0.08, 0.7)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)
4r-u_shape_blocked_a-unlock_doors	0.0 (0.0, 0.0)	0.36 (0.06, 0.66)	1.0 (1.0, 1.0)	0.92 (0.68, 1.0)
6r-corridors-arrange_keys	0.0 (0.0, 0.0)	0.22 (0.04, 0.46)	0.52 (0.34, 0.68)	0.64 (0.42, 0.86)
6r-corridors-block_doors	0.0 (0.0, 0.0)	0.42 (0.3, 0.52)	0.74 (0.5, 0.88)	0.5 (0.25, 0.68)
6r-corridors-find	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)
6r-corridors-green_square_search	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.18 (0.1, 0.26)	0.0 (0.0, 0.0)
6r-key_objective-find_red_ball	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-key_objective-key_tour	0.0 (0.0, 0.0)	0.12 (0.06, 0.18)	0.62 (0.42, 0.84)	0.48 (0.32, 0.72)
6r-key_objective-pair_balls	0.0 (0.0, 0.0)	0.92 (0.68, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-key_objective-return_to_base	0.0 (0.0, 0.0)	0.8 (0.28, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
6r-key_temple-ball_offer	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.12 (0.06, 0.18)	0.04 (0.0, 0.16)
6r-key_temple-pair_chain	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.52 (0.36, 0.7)	0.04 (0.0, 0.16)
6r-key_temple-ritual	0.0 (0.0, 0.0)	0.06 (0.02, 0.1)	0.56 (0.14, 0.72)	0.06 (0.0, 0.16)
6r-key_temple-unlock_chain	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.16 (0.04, 0.28)	0.0 (0.0, 0.0)
6r-locked_rooms-ball_sequence	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.7 (0.46, 0.94)	0.08 (0.0, 0.24)
6r-locked_rooms-find_blue_ball	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-locked_rooms-green_stuff	0.0 (0.0, 0.0)	0.96 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-locked_rooms-unlock_doors	0.0 (0.0, 0.0)	0.36 (0.24, 0.6)	1.0 (1.0, 1.0)	0.62 (0.48, 0.88)
6r-secret_treasure-find_treasure	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)
6r-secret_treasure-pairings	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)
6r-secret_treasure-take_the_key	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
6r-secret_treasure-unlock_all	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.04 (0.0, 0.16)	0.02 (0.0, 0.08)
anyorder_2o	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
anyorder_3o	0.0 (0.0, 0.0)	0.9 (0.84, 0.96)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_four-carrying_keys	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	0.96 (0.92, 1.0)

Table 7: Solve rates and 95% CIs (in brackets) for hand-designed problems with *random walk-based* task sampling (continued).

Problem	DR_{indep}	$PLR^{\perp}_{indep}$	DR_{cond}	$PLR^{\perp}_{cond}$
choice-choice_four-front_green_square	0.0 (0.0, 0.0)	0.18 (0.12, 0.2)	0.66 (0.54, 0.82)	0.42 (0.28, 0.76)
choice-choice_four-front_red_square	0.0 (0.0, 0.0)	0.8 (0.36, 1.0)	1.0 (1.0, 1.0)	0.82 (0.74, 0.88)
choice-choice_four-no_unlocking	0.0 (0.0, 0.0)	0.14 (0.06, 0.24)	0.4 (0.1, 0.68)	0.34 (0.14, 0.62)
choice-choice_four-pair_blue_red_objs	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.08 (0.02, 0.14)	0.0 (0.0, 0.0)
choice-choice_four-patience_test	0.0 (0.0, 0.0)	0.62 (0.46, 0.8)	0.8 (0.72, 0.92)	0.68 (0.38, 0.88)
choice-choice_four-red_ball_square	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.04 (0.0, 0.08)	0.04 (0.0, 0.08)
choice-choice_four-two_red_squares	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.62 (0.48, 0.74)	0.08 (0.02, 0.16)
choice-choice_one-arrange_squares	0.0 (0.0, 0.0)	0.82 (0.28, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-key_to_square	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-move_everything	0.0 (0.0, 0.0)	0.96 (0.84, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-retrieve_square	0.0 (0.0, 0.0)	0.9 (0.78, 0.98)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
choice-choice_one-toggle_door_squares	0.0 (0.0, 0.0)	0.74 (0.42, 1.0)	0.98 (0.92, 1.0)	0.96 (0.84, 1.0)
choice-choice_one-unlock_door	0.36 (0.12, 0.72)	0.96 (0.84, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-unlock_door_specific	0.02 (0.0, 0.08)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_one-visit_all_squares	0.0 (0.0, 0.0)	0.9 (0.6, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
choice-choice_three-front_ball_blue	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.38 (0.2, 0.56)	0.06 (0.0, 0.16)
choice-choice_three-front_ball_purple	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.4 (0.28, 0.48)	0.0 (0.0, 0.0)
choice-choice_three-next_to_square	0.0 (0.0, 0.0)	0.7 (0.44, 0.88)	0.96 (0.92, 1.0)	0.86 (0.76, 0.96)
choice-choice_three-next_to_square_a	0.0 (0.0, 0.0)	0.8 (0.42, 0.94)	0.66 (0.56, 0.74)	0.9 (0.82, 0.98)
choice-choice_three-next_to_square_b	0.0 (0.0, 0.0)	0.06 (0.02, 0.1)	0.38 (0.24, 0.52)	0.26 (0.1, 0.42)
choice-choice_three-next_to_squares	0.0 (0.0, 0.0)	0.6 (0.38, 0.88)	0.76 (0.54, 0.9)	0.78 (0.6, 0.96)
choice-choice_three-next_to_squares_a	0.0 (0.0, 0.0)	0.22 (0.14, 0.3)	0.72 (0.64, 0.84)	0.2 (0.06, 0.34)
choice-choice_three-unlocking_sequence	0.0 (0.0, 0.0)	0.26 (0.08, 0.36)	0.88 (0.76, 0.96)	0.32 (0.1, 0.8)
choice-choice_two-avoid_unlock	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.06 (0.0, 0.16)	0.0 (0.0, 0.0)
choice-choice_two-avoid_unlock_hard	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.04 (0.0, 0.08)	0.0 (0.0, 0.0)
choice-choice_two-avoid_unlock_impossible	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)
choice-choice_two-front_square_green	0.0 (0.0, 0.0)	0.22 (0.08, 0.38)	0.92 (0.84, 0.98)	0.54 (0.36, 0.83)
choice-choice_two-front_squares	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.4 (0.22, 0.6)	0.14 (0.02, 0.26)
choice-choice_two-front_squares_hard	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.46 (0.32, 0.62)	0.18 (0.06, 0.3)
choice-choice_two-intermediate_door_unlock	0.0 (0.0, 0.0)	0.64 (0.2, 0.88)	0.96 (0.84, 1.0)	0.72 (0.48, 0.88)
choice-choice_two-next_key_square	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.5 (0.32, 0.8)	0.04 (0.0, 0.16)
choice-choice_two-next_to_squares_0	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.38 (0.2, 0.54)	0.02 (0.0, 0.08)
choice-choice_two-next_to_squares_1	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.34 (0.16, 0.56)	0.04 (0.0, 0.08)
myopic	0.0 (0.0, 0.0)	0.58 (0.36, 0.72)	0.48 (0.4, 0.64)	0.48 (0.38, 0.54)
patrol_4r_full_spec	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.86 (0.76, 0.94)	0.14 (0.0, 0.56)
patrol_4r_overspec	0.0 (0.0, 0.0)	0.56 (0.18, 0.88)	0.96 (0.92, 1.0)	0.74 (0.5, 0.96)
patrol_4r_underspec	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.7 (0.58, 0.88)	0.0 (0.0, 0.0)
patrol_4r_underspec-1	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.8 (0.68, 0.92)	0.04 (0.0, 0.16)
patrol_6r	0.0 (0.0, 0.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
patrol_6r_objects	0.0 (0.0, 0.0)	0.84 (0.68, 0.96)	1.0 (1.0, 1.0)	0.96 (0.92, 1.0)
patrol_6r_objects_obstacles	0.0 (0.0, 0.0)	0.36 (0.12, 0.62)	0.94 (0.76, 1.0)	0.92 (0.84, 0.98)
patrol_6r_obstacles	0.0 (0.0, 0.0)	0.88 (0.68, 1.0)	1.0 (1.0, 1.0)	0.98 (0.92, 1.0)
putnext_1r	0.0 (0.0, 0.0)	0.68 (0.54, 0.88)	1.0 (1.0, 1.0)	0.96 (0.84, 1.0)
putnext_2r-0	0.0 (0.0, 0.0)	0.92 (0.76, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
putnext_2r-1	0.0 (0.0, 0.0)	0.9 (0.78, 0.98)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
putnext_2r_distractor-0	0.0 (0.0, 0.0)	0.08 (0.02, 0.16)	0.56 (0.44, 0.78)	0.42 (0.33, 0.54)
putnext_2r_distractor-1	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.38 (0.24, 0.52)	0.2 (0.04, 0.4)
putnext_4r	0.0 (0.0, 0.0)	0.98 (0.92, 1.0)	1.0 (1.0, 1.0)	1.0 (1.0, 1.0)
unlocktounlock_overspec	0.0 (0.0, 0.0)	0.8 (0.28, 0.98)	1.0 (1.0, 1.0)	0.96 (0.84, 1.0)
unlocktounlock_underspec	0.0 (0.0, 0.0)	0.28 (0.06, 0.72)	0.92 (0.84, 0.98)	0.8 (0.36, 0.96)
walls-chunks-blue_ball	0.0 (0.0, 0.0)	0.9 (0.68, 1.0)	0.5 (0.26, 0.74)	0.82 (0.68, 0.92)
walls-chunks-green_ball	0.08 (0.02, 0.16)	0.82 (0.68, 0.94)	0.38 (0.24, 0.68)	0.8 (0.5, 0.92)
walls-chunks-grey_ball	0.18 (0.0, 0.72)	0.9 (0.82, 0.98)	0.64 (0.5, 0.9)	0.92 (0.84, 0.98)
walls-chunks-purple_ball	0.0 (0.0, 0.0)	0.7 (0.4, 0.82)	0.32 (0.1, 0.58)	0.82 (0.62, 0.96)
walls-chunks-red_ball	0.04 (0.0, 0.08)	0.06 (0.0, 0.16)	0.08 (0.02, 0.14)	0.02 (0.0, 0.08)
walls-chunks-yellow_ball	0.14 (0.0, 0.4)	0.36 (0.1, 0.5)	0.1 (0.04, 0.16)	0.16 (0.06, 0.32)
walls-maze_4r-red_ball	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)
walls-maze_4r-red_key	0.06 (0.0, 0.24)	0.62 (0.4, 0.8)	0.28 (0.14, 0.42)	0.2 (0.0, 0.8)
walls-maze_4r-red_square	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)
walls-maze_6r-green_ball	0.0 (0.0, 0.0)	0.36 (0.0, 0.72)	0.02 (0.0, 0.08)	0.02 (0.0, 0.08)
walls-maze_6r-green_key	0.0 (0.0, 0.0)	0.04 (0.0, 0.16)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)
walls-maze_6r-green_square	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)
walls-maze_9r-blue_ball	0.0 (0.0, 0.0)	0.2 (0.06, 0.38)	0.0 (0.0, 0.0)	0.12 (0.0, 0.32)
walls-maze_9r-blue_key	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)
walls-maze_9r-blue_square	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)
walls-spiral_4r-purple_key	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.02 (0.0, 0.08)
walls-spiral_9r-grey_key	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)	0.0 (0.0, 0.0)