Leveraging VLMs for Specification-Guided Policy Learning

Anonymous submission

Abstract

Recent work has demonstrated how formal specifications can guide reinforcement learning (RL) agents to successfully solve long-horizon tasks. However, existing approaches require high-quality human-designed specifications. Our main insight is that vision-language foundation models (VLMs) can help guide specification design. We propose an algorithm that starts from an initial user-provided specification and leverages a VLM to iteratively refine it to improve the performance of the RL agent. A key challenge is that the VLM has limited understanding of how refinements affect the RL agent's success. Thus, we incorporate symbolic refinements that "tune" the VLM-proposed refinements to significantly improve their efficacy. In experiments on navigation tasks, we demonstrate that our framework produces substantially better specifications compared to both the initial human specification and a one-shot VLM refinement. In particular, our method scales well to long-horizon tasks, significantly outperforming the VLM and symbolic baselines.

1 Introduction

A key challenge in reinforcement learning is the *specification problem*—i.e., how the user describes the task to the agent. A promising approach is to specify tasks using *task graphs*, often expressible in a temporal logic (Jothimurugan, Alur, and Bastani 2019). This approach encodes high-level user intent transparently and interpretably (compared to reward functions), is unambiguous (compared to natural language), and can be formally analyzed for correctness guarantees (Rozier 2011). Further, task graphs decompose naturally into subproblems which can be solved modularly. This aligns with conventional notions of good software design, and has so far been helpful when using deep learning methods to solve long-horizon tasks (Jothimurugan et al. 2021).

Additionally, prior work has proposed leveraging the structure of the specification to improve learning (Jothimurugan et al. 2021). Their algorithm trains separate policies to solve each sub-goal, and then composes these policies together to solve the overall task. To ensure efficiency, they use Dijkstra's algorithm to guide high-level decisions about which sub-goals to solve next, and then use off-the-shelf reinforcement learning algorithms to solve each one.

Notably, in this paradigm the task graph embodies two disparate roles: it is a *specification* of the requirements for achieving the task, while also being an *implementation* of the structure of a compositional policy which aims to solve the task. These two functionalities are not necessarily aligned; for instance, a particular valid specification might not sufficiently decompose the task into primitive subtasks that can be solved efficiently by reinforcement learning. On the other hand, a more restrictive specification might generate a high-quality compositional RL controller, but an *overly* restrictive specification may also be difficult to satisfy, and might not adequately express the user's intent. Therefore, when specifying a task graph, a designer benefits from considering not only the true task specification but also the lower-level details of the environment. This gives users more control over the learning algorithm, enabling them to improve its performance by modifying the specification. However, due to the indirect nature of the task graph's influence on the learning algorithm, non-experts may not be aware of how to realize these gains.

One solution is to automatically *refine* an initial user specification to make it more amenable to learning (Ambadkar, Žikelić, and Verma 2025); for instance, decompose long-term goals by adding intermediate subgoals, or improve the precision of a goal to help the agent avoid low-quality states. With automatic refinement, the user gets the benefits of a refined specification without the expert knowledge required to create one themselves. However, existing specification refinement algorithms rely purely on classical algorithms, which have limited ability to reason about complex environment dynamics, so may produce refinements that are not achievable by the agent or not useful for completing the task.

Alternatively, foundation models have recently demonstrate advanced abilities at code comprehension, code writing, and complex reasoning. Large language models (LLMs) and vision-language models (VLMs) have been used to generate waypoints, formal task specifications, and reward functions from natural language and/or source code (Yu et al. 2023; Ma et al. 2023; Wang et al. 2024b,a; Liang et al. 2024). However, in our experiments, we find that these approaches alone are unable to produce effective specification refinements. Intuitively, effective specification refinements require

not only high-level task structure (which can be expressed to the VLM), but also low-level understanding of the dynamics model (which is difficult to express to the VLM).

Contributions. We propose a novel strategy that combines vision-language models (VLMs) with symbolic reasoning to perform more effective specification refinement. Intuitively, the VLM proposes high-level refinements that are subsequently tuned by symbolic reasoning to ensure their efficacy. In more detail, our method combines symbolic search over the task graph with VLM-proposed refinements, producing a specification that is much more effective at guiding the agent to its goal. We evaluate our proposed method against 3 benchmarks on 2D navigation tasks with the MuJoCo Ant, demonstrating that our method consistently outperforms unscaffolded VLM task graph refinement, with performance gains of at least $2\times$ across tasks with multiple rooms.

2 Overview

Consider the RL environment represented in Figure 1. In this task, an agent (Towers et al. (2024)'s MuJoCo Ant) is initialized in the initial state set (the cyan region) at a uniform random point, and must complete a reach-avoid task by navigating to the goal state set (green region) while avoiding the walls (black regions). The observation space is continuous with 29 dimensions (e.g., joint positions and velocities); here, we render only 2 of the position dimensions to represent the navigation task.

Formally, this RL problem is defined over a Markov decision process (MDP) $\mathcal{M} = (S, A, P, R)$. Here, S is the state space, A is the action space, $P : S \times A \times S \to [0, 1]$ is the state transition function, and $R : S \times A \times S \to \mathbb{R}$ is the reward function. A learned stochastic policy $\pi : S \times A \to [0, 1]$ maps states to distributions over the action space.

A pure RL solution could apply a learning algorithm, such as Proximal Policy Optimization (PPO), to learn an end-to-end neural network policy that completes the task. Under this approach, the neural net must encode both the low-level dynamics of the ant environment and the higher-level task of navigating this specific maze. This has a major drawback: given a different maze for the same agent, a new RL policy would need to be learned.

Our solution separates these concerns. We represent the high-level navigation challenge in a task graph. Separately, we use RL to pretrain a goal-conditioned neural policy that is responsible for learning the low-level environment dynamics. Importantly, the task graph can be arbitrarily modified without requiring any retraining or finetuning of the neural network policy. This also simplifies the RL problem to just environment dynamics, allowing the neural policy to be learned more quickly and with fewer parameters. The task graph and subtask policies are combined to form a compositional policy that solves the task.

In general, a task graph $G = (V, E)$ represents a control task abstractly. G is directed and acyclic, with vertex set V and edge set E. Each vertex $v \in V$ corresponds to a subset $I(v) \subseteq S$ of member states in the base problem. $v_0 \in V$ denotes the initial vertex, and $F \subseteq V$ denotes the final vertices. Each edge $e = (u, v) \in E$ corresponds to a subset $O(e)$ of unsafe states in the base problem. Thus, a trajectory of the agent in the environment is said to successfully complete the subtask e if it starts in $I(u)$ and reaches $I(v)$ while avoiding $O(e)$. The *weight* of edge $u \xrightarrow{e} v$ is equal to the negative log-probability that the agent accomplishes subtask, conditional on starting in $I(u)$. Many formal specification languages exist for the expression of task graphs, such as SpectRL (Jothimurugan, Alur, and Bastani 2019), which expresses sequential reach-avoid tasks. The application of task graphs to RL problems is **compositional reinforcement learning** (Andreas, Klein, and Levine 2017).

Worked example. The initial task graph for the problem in Figure 1 consists of a single directed edge, which encodes the reach-avoid task specification. The initial state set is represented by the source vertex v_0, while the goal region is sink vertex v_1. The first 3 steps of our procedure are:

First, the end-to-end task is evaluated in simulation. The low success rate ($p = 0.5$) of the trajectories demonstrates that the goal-conditioned RL policy cannot effectively satisfy the specification.

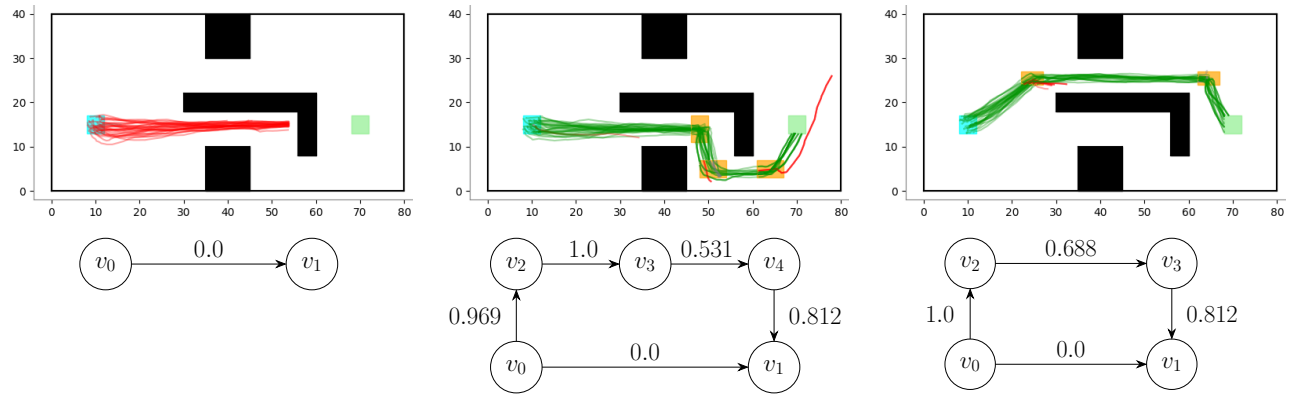
Second, this evaluation data is provided to a VLM, which returns a sample of candidate refinements for the edge (this step is detailed in Algorithm 1). Each candidate refinement is a path meant to replace the existing edge. For instance, Figure 1 (c) proposes to break the task into a sequence of 3 subtasks: the reach-avoid task $v_0 \to v_2$, followed by $v_2 \to v_3$ and finally $v_3 \to v_1$.

Third, each of these two candidate refinements is evaluated in simulation, and the probability of successfully satisfying the refined specification is measured. The candidate refinement in Figure 1 (c) performs better, with success rate $55.9\% > 41.8\%$, so it is integrated into the task graph.

At 55.9%, this new compositional controller still does not achieve a satisfactory rate of success for the *original* reach-avoid task. Therefore the procedure continues to iterate over further refinements before finally returning a solution.

Refinement procedure. We propose a specification refinement procedure that performs an A* search over the task graph. The overall search-and-refinement loop is illustrated in Figure 2. Within this loop, graph elements in the open set of the search are subject to mutation using two refinement strategies. **Edge refinements**, as demonstrated in the illustrative worked example above (and in Algorithm 1), replace an edge with a sequence of smaller subtasks. **Vertex refinements** (Algorithm 2, Figure 3), replace the membership set $I(v)$ of a vertex v with a strict subset of its states. In other words, edge refinement introduces subtasks to break up a monolithic task, while vertex refinement makes a goal more precise to improve learning. On top of this, the graph structure captures branching decisions that must be made based on the complex dynamics of the task (e.g., the two doorways in Figure 1). A* search efficiently schedules subtasks to be evaluated and refined, and ensures that the final compositional policy is optimal for the generated task graph.

Experiments. We focus on 2D navigation tasks, where an agent must navigate a maze to reach its final goal. This domain is well-suited to reach-avoid task graph specifications. For VLMs to generate high-quality candidate edge refinements, we must construct a prompt that includes all relevant



(a) Original input task graph (single edge), with 0 of 32 successful trajectories. Initial vertex is centered at $(10, 15)$ in cyan. Final vertex is at $(70, 15)$ in green. Avoid regions (walls) are black.

(b) Candidate edge refinement with VLM-proposed subgoals in orange. The agent's empirical probability of successfully traversing the path is 41.8%.

(c) Alternative edge refinement. The agent's empirical probability of successfully traversing the path is 55.9%.

Figure 1: Split doorway task in the MuJoCo Ant environment using a pretrained neural network policy to evaluate each subtask. Successful trajectories are rendered in green, while unsuccessful ones are red. Experiment setup is detailed in Section 5.

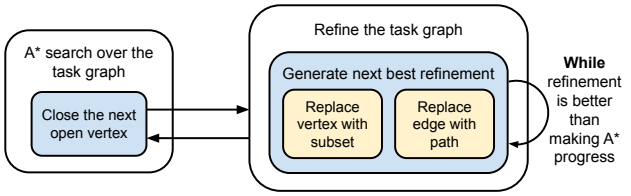


Figure 2: Our algorithm for task graph refinement iterates between A*-based iterative evaluation of subtasks (left) and local mutations of graph components (right).

information needed to select effective refinements. While generalizable, these contexts must be tailored to the specific domain. In the 2D navigation domain, information can be visualized by rendering the initial position, goal location, and trajectory attempts onto a 2D map of the environment. This way, the VLM observes useful information about how the dynamics prevents the agent from achieving its goal.

We experimentally evaluate our approach at specification refinement on a 2D navigation task where the agent (a MuJoCo ant) must traverse a maze to reach a goal. Our iterative refinement strategy successfully generates sparse task graphs whose induced compositional policies exhibit consistently higher success rates than VLM refinement alone.

3 Related Work

Specification-guided RL. In specification-guided RL, the task is given in the form of a logical specification, where the logical primitives represent the abstract actions (Jothimurugan, Alur, and Bastani 2019; Icarte et al. 2022; Vaezipoor et al. 2021; Zhang et al. 2024; Jackermeier and Abate 2024). The abstract decision problem induced by the task graph in the higher-level abstraction space is typically assumed to be Markov. However, this assumption is not upheld in many examples to which we would like to apply these methods (Jothimurugan, Bastani, and Alur 2021). This causes a disconnect between the high- and low-level policies, preventing the agent from learning a consistently high-performing hierarchical policy. The divided doorway task is one example of this, shown in Figure 1.

LLM-guided control. LLMs have impressive code generation capabilities. Ma et al. (2023) presented Eureka, which generated shaped reward functions for RL policies on manipulation tasks. The framework used evolutionary LLM prompting, interleaved with RL training to evaluate each generation of candidate rewards. However, this was limited to short-horizon fine motor tasks. Reasoning in long-horizon, complex tasks remains a challenge for LLMs and VLMs.

Foundation models including LLMs and VLMs have been used for robotic path planning in the navigation domain (Cao et al. 2024; Shah et al. 2023). However, these methods often rely on domain-specific data or training, and do not produce interpretable policies that can be rigorously checked. These limitations mean that they cannot be applied in new, safety-critical domains.

Hierarchical RL. The field of hierarchical reinforcement learning (HRL) aims to scaffold RL problems by learning hierarchies of policies over abstract actions. For example, to solve the task of navigating through a maze of rooms, the agent might learn RL policies for opening doors and navigating each room as independent subtasks, or options (Sutton, Precup, and Singh 1999), over the primitive actions, and then a higher-level RL policy over these abstract actions. The joint hierarchical policy then solves the task.

Much of this work assumes that the abstract actions are given and fixed (Pateria et al. 2021). This is partly because it enables all of the reward functions in the problem to be hand-designed. Automatically generated or universal reward functions that are also well-shaped (not sparse, thus suitable for learning) using classical methods is challenging. Without

them, it is impossible to construct new sub-problems on the fly in the training loop, which means that automatically constructing and modifying RL problem decompositions has only been successful in limited cases (Stolle and Precup 2002; Ramesh, Tomar, and Ravindran 2019; Kim, Park, and Kim 2021).

4 Neurosymbolic Task Graph Refinement

Our method takes as input a directed acyclic task graph $G = (V, E)$. Each edge defines a reach-avoid subtask. Subtask policies can take any form: model-predictive controller, pretrained neural network policy, or RL policy trained or fine-tuned in-the-loop. Edge weights are equal to the negative log-probability of the chosen subtask policy satisfying the subtask. Given this structure, learning an optimal compositional policy is equivalent to solving a single-source shortest path (SSSP) problem.

We execute an A* search (Hart, Nilsson, and Raphael 1968) over the task DAG. Edge weights (and thus, simulations) are evaluated lazily to reduce computation costs. We iteratively modify the DAG in-place by interleaving graph *refinements* with the progression of the A* algorithm. These refinements consist of traditional symbolic logic programs (Algorithm 2) and external queries to off-the-shelf pretrained foundation models (Algorithm 1), and mutate localized subgraphs in the DAG. They increase the structural information contained in the task graph, simplifying subtasks and thus decreasing distances between vertices.

This procedure discovers the refined graph's shortest path, which corresponds to a composite policy comprising a sequence of consecutive subtasks guaranteed to satisfy the original task specification.

4.1 Graph Refinements

Our procedure applies two types of localized graph refinement: edge and vertex refinements.

Edge split refinements, described in Algorithm 1, require query access to a Vision-Language Model (VLM), referred to as variable $\mathcal{M}$. They are also parameterized by a number of iterations d and number of samples w per iteration. Given input edge $e \in E$, the VLM is queried for w candidate refinements. Results from in-simulation evaluations of edge e are provided: the empirical probability of successfully completing the subtask, and a 2D image rendering of the evaluated trajectories. Also included in the context are source code for the simulator environment analogous to Ma et al. (2023), and the specification language API. Each proposed refinement, $[e^j_{i=1,\dots,k_j}]$ for $j = 1, \dots, w$, decomposes the edge into a sequence of subtasks. Thus, if $u \xrightarrow{e} v$, we have:

$$u \xrightarrow{e^j_1} x_1 \xrightarrow{e^j_2} \dots \xrightarrow{e^j_{k_j-1}} x_{k_j-1} \xrightarrow{e^j_{k_j}} v$$

Each of the w such sequences is evaluated in the environment to estimate the probability of successfully completing it; the same numerical and rendering feedback are included in the next iterative prompt. Finally, the best-performing refinement of the $d \times w$ candidates is selected. The illustrative example presented in Section 2 and Figure 1 depicts an edge refinement according to this procedure. Details of the VLM prompts are provided in our Supplemental Materials.

Algorithm 1: Generate Edge Refinement

Input: Edge $u \xrightarrow{e} v$; integers w, d; foundation model $\mathcal{M}$
Output: Candidate partition of edge e into $[e_{i=1\dots k}]$
Output: Subtask policy and success prob of e_i
$S \leftarrow \mathcal{M}.\texttt{query}(e, \text{samples} = w)$
$\vec{\pi} \leftarrow \left[\left[\text{policy } \pi^j_i \text{ for } e^j_i \text{ in } \vec{e}_j\right] \text{ for } \vec{e}_j = \left[e^j_{i=1\dots k_j}\right] \text{ in } S\right]$
$\vec{p} \leftarrow \left[\left(\prod_{i=1}^{k_j} \text{success probability of } \pi^j_i\right) \text{ for } \vec{e}_j \text{ in } S\right]$
$p_{\text{best}},\ \vec{e}_{\text{best}},\ \vec{\pi}_{\text{best}} \leftarrow \max(\vec{p}),\ \arg\max(\vec{p}),\ \vec{\pi}[\vec{e}_{\text{best}}]$
for $t = 1 \dots d-1$ **do**
  $S \leftarrow \mathcal{M}.\texttt{query}(e, S, \vec{\pi}, \text{samples} = w)$
  Apply Algorithm 2 to each path in S
  $\vec{\pi} \leftarrow \left[[\text{policy } \pi^j_i \text{ for } e^j_i \text{ in } \vec{e}_j] \text{ for } \vec{e}_j = [e^j_{i=1\dots k_j}] \text{ in } S\right]$
  $\vec{p} \leftarrow \left[\left(\prod_{i=1}^{k_j} \text{success probability of } \pi^j_i\right) \text{ for } \vec{e}_j \text{ in } S\right]$
  $p_{\text{best}},\ \vec{e}_{\text{best}} \leftarrow \max(p_{\text{best}}; \vec{p}),\ \arg\max(\vec{e}_{\text{best}}; \vec{p})$
  $\vec{\pi}_{\text{best}} \leftarrow \vec{\pi}[\vec{e}_{\text{best}}]$
end for
output Subtasks $\vec{e}_{\text{best}}$ and policies $\vec{\pi}_{\text{best}}$

Vertex subset refinements are applicable to any chain of vertices $v_1, \dots, v_\ell$ such that $\deg_{in}(v_i) = \deg_{out}(v_i) = 1\ \forall i$. We present the special case $\ell = 1$ in Algorithm 2. Internally, $O(2k)$ subtasks are evaluated.

Algorithm 2: Generate Vertex Refinement

Input: Vertex v such that $\deg_{\text{in}}(v) = \deg_{\text{out}}(v) = 1$, so we have neighboring edges $u \xrightarrow{e_1} v \xrightarrow{e_2} w$; integer k
Output: Candidate subset $I'(v)$ of $I(v)$
Output: Subtask policy and success prob of e'_1 and e'_2, where $I(u) \overset{e'_1}{\rightsquigarrow} I'(v) \overset{e'_2}{\rightsquigarrow} I(w)$
Sample k sets of states $s_1, \dots, s_k \subset I(v) = s_0$
for $i = 0 \dots k$ **do**
  Evaluate policies for $I(u) \rightsquigarrow s_i$ and $s_i \rightsquigarrow I(w)$
  $p_i \leftarrow \Pr[s_i \mid I(u)] \cdot \Pr[I(w) \mid s_i]$
end for
output State set s_i, policies, and weights for $i = \arg\min_i p_i$

The approach also extends from the single input vertex to a sequence of vertices $x_1, \dots, x_t$ produced in the edge refinement module. For such inputs, the vertex refinement evaluates $O(2k + k^2(t-1))$ subtasks in order to optimize over $O(k^{t+1})$ possible paths from u to w, using dynamic programming to iterate over the layers of the sequence.

Generalizability. While our evaluation implements these two refinement procedures, in principle, any mutation rule that replaces a task subgraph with a refined subgraph could be integrated into the strategy we propose. A new task domain might motivate the implementation of a different library of graph refinements. The primary requirement for such a rule is that the replacement subgraph equals a formal specification that implies the original.

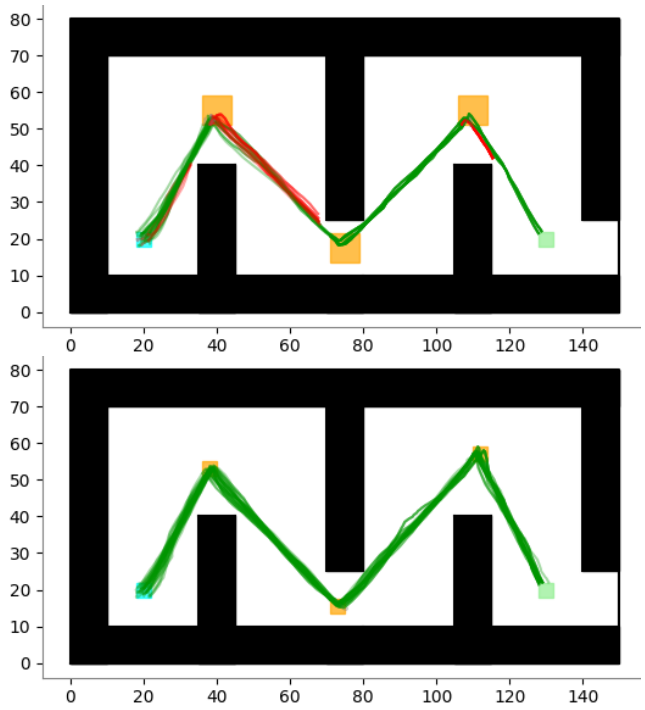


Figure 3: Task defined by composing two U Maze rooms. The input task graph is a single edge, with initial vertex in cyan (leftmost) and final vertex in green (rightmost). A candidate Edge Refinement is first produced by the VLM (upper image). The new edges' success probabilities are $(0.812,\ 0.312,\ 0.875,\ 0.406)$, so the overall task probability is 9%. Next, we apply vertex refinement (Algorithm 2) along the path, producing the lower image with success probabilities $(1.0,\ 1.0,\ 0.969,\ 1.0)$ for a total of 96.9%.

4.2 Iterative Exploration

Throughout its runtime, the procedure maintains a priority ranking of candidate graph refinements as generated by Algorithms 1 and 2. Allowing any graph component to be mutable at any point in the procedure would lead to unbounded graph complexity and runtime; instead, our procedure considers a restricted set of refinements (the data structure `refinements` in Algorithm 3). This set evolves as the graph is explored, guaranteeing that the procedure makes continual progress through the graph, and is naturally induced by the state of the A* search. The `closed` set of the A* search defines a gradually expanding subgraph which is considered finalized and not eligible for further graph mutation. The frontier, or `open` set, contains the graph components over which the `refinements` priority queue is built (Algorithm 3, line 10) and maintained (line 21).

Concretely, the `open` set consists of vertices $v \in V$ for which a current-best path from the source is maintained, but not yet known to be minimal. This is exactly the set of vertices for which vertex refinements via Algorithm 2 are considered. Every such vertex (except for the source itself, at the beginning of the procedure) has an associated preceding edge along its current-best path. This is exactly the set of edges for which edge refinements via Algorithm 1 are considered. All refinements can apply, recursively, to new graph components constructed by previous refinements. Algorithm 3 shows the overall procedure, which interleaves A* search with graph refinement.

Algorithm 3: Evaluate and Refine

1: **Input:** Task graph $G = (V, E)$
2: **Input:** Initial vertex $v_0 \in V$, final vertices $F \subseteq V$, and initial distribution $\eta_0 : I(v_0) \to [0, 1]$
3: **Input:** Rollout access to the task MDP, given edge $e \in E$
4: **Input:** $\tau > 0$, integers w, d, k, and foundation model $\mathcal{M}$

5: **Output:** Task graph $G' = (V', E') \geq G$
6: **Output:** Inclusion map I' over V', refined from I on V
7: **Output:** Family of policies $\pi : E' \times S \to A$

8: `closed` $\leftarrow \emptyset$
9: Priority queue `open` $\leftarrow \{\langle v_0,\ 0.0\rangle\}$
10: Build priority queue `refinements`

11: **while** `open` is nonempty **do**
12: {A* progress}
13: Move lowest-cost `open` vertex u to `closed`
14: **if** $u \in F$ **then** Break
15: **for** edge $u \xrightarrow{e} v$ **do**
16: Evaluate subtask policy π_e
17: If no better path is already known, add v to `open` with priority `cost`(u) + `weight`(e)
18: **end for**
19: {Refine}
20: **repeat**
21: Update `refinements`
22: $c_{\text{progress}} \leftarrow$ `cost(open.peek())`
23: $c_{\text{refine}} \leftarrow$ `cost(refinements.peek())`
24: **if** $c_{\text{refine}} < c_{\text{progress}} - \tau$ **then**
25: Incorporate `refinements.pop()` into G
26: Update `open`
27: **end if**
28: **until** $c_{\text{progress}} < c_{\text{refine}} + \tau$
29: **end while**
30:
output Refined task graph G', subtask policies π

4.3 Validity and Optimality Guarantees

Theorem 4.1 (Task graph validity). For input task graph G and refined task graph G' output by our procedure, for any trajectory τ, if τ satisfies the specification given by G' then it must also satisfy the specification given by G.

Proof. It suffices to observe that each local task graph refinement (Algorithms 1 and 2) obeys this implication. For `vertex_subset`(v) with $v \in V'$, we have that $I'(v) \subseteq I(v)$ per the definition of the refinement. Therefore wherever τ reaches v in G' via $I'(v)$, it also reaches v in G via $I(v)$. For $(e_1 \to \cdots \to e_k) =$ `edge_split`$(e_{u\to v}, w, d, \mathcal{M})$ with $u, v \in V'$ and $e_{u\to v} \in E'$, the new edges inherit

the avoid region $O(e_{u\to v})$ of the original edge. Therefore for any contiguous subsequence of states in τ that satisfies $e_1 \to \cdots \to e_k$, the initial state must satisfy u (as e_1 starts at u), the final state must satisfy v (as e_k ends at v) and the intermediate states must all avoid $O(e_{u\to v})$. Thus the subsequence satisfies e. □

Theorem 4.2 (Compositional policy optimality). The output compositional policy is optimal for the output task graph G'.

Proof. In Algorithm 3, the compositional policy is defined by tracing back source vertices from the final `open` vertex u. This corresponds to the shortest path result in a standard SSSP A* search. Observe that the search progress component of Algorithm 3 matches the logic of the A* algorithm with an inadmissible heuristic function. The search algorithm only accesses graph components and weights as they are reached by the `open` set, while refinements may only decrease distances in the graph, and may only affect graph components outside of the subgraph induced by the `closed` set. Thus if the algorithm's termination in finite time is assumed, then by correctness of the A* search, the claim holds. □

5 Experiments

Environment. We evaluate our method using a custom version of the Gymnasium-Robotics (de Lazcano et al. 2024) AntMaze environment, implemented on top of the MuJoCo Ant (Towers et al. 2024; Todorov, Erez, and Tassa 2012). The environment supports 29-dimensional observations and 8-dimensional actions, both continuous spaces. 2D image rendering depicts the (x, y) coordinates of agents and maze walls. For subtask learning, the state space is goal-augmented using a representative waypoint randomly sampled from the inclusion set of the subtask's goal vertex. For rendered evaluations of example tasks, see Figures 1 and 3.

Tasks. Our suite of benchmark tasks is based on the evaluation reported in Jothimurugan et al. (2021). We start with a core set of one-room mazes, including the Doorway task shown in Figure 1.

To assess performance under scaling task complexity, we define compositional rooms by chaining together copies of the "X Maze" and "U Maze," respectively. An example is the 2-room U Maze, rendered in Figure 3. Notably, we consider two types of task graphs for the compositional U Maze environments. Under one paradigm, the inital task graph stays constant with one edge describing the start-to-finish task ("no subgoals"). Under the second paradigm, the size of the task graph grows linearly with the number of rooms in the compositional task: subgoals are placed at the doorways between rooms, so the initial task graph is a path that scales with task complexity ("with subgoals"). This structure provided by the initial task graph means that the residual difficulty of the task should remain relatively constant, even as complexity of the start-to-finish task increases.

Hyperparameters. We implement Edge Refinement (Algorithm 1) with $w = 3$ candidate refinements sampled per iteration and $d = 2$ iterations per call to the module. We implement Vertex Refinement (Algorithm 2) with $k = 5$ subsets evaluated for a given vertex. We evaluate 32 rollouts to estimate the success rate of a subtask. The tolerance parameter τ is on a simple linear scaler, starting at 0.02 and capped at 0.1. Foundation model queries use OpenAI's GPT 5.6 Terra (OpenAI 2025) or Google's Gemini 3.5 Flash (Team et al. 2023), with hyperparameters (thinking, temperature, etc) at their defaults. All VLM prompts are provided in Suppelementary Materials.

Subtask control policy. The subtask policy is a neural network policy of 2×256 weights, pretrained by Proximal Policy Optimization (PPO, Schulman et al. (2017)) on a goal-conditioned reach task in the Ant environment.

5.1 Baselines

We compare against 3 baseline methods. All use the same subtask policy (a pretrained goal-conditioned neural network) and hyperparameters, where applicable:

VLM baseline. While our approach refines individual graph components iteratively, this baseline takes a one-shot approach, using similar prompts to the VLM for a complete refinement of the input task graph. Context includes a one-shot example refinement with explained reasoning. Like our Edge Refinements, there is a second iteration of feedback: VLM-proposed are evaluated in the simulation, and results are fed back to the VLM for a follow-up query. The best candidate of the 2×3 proposed task graphs is selected. Unlike our method, there is no formal guarantee (akin to Theorem 4.1) that the task graph produced is a true refinement of the original specification. However, the optimality of the compositional policy (Theorem 4.2) applies.

Symbolic refinements baseline. This is just our solution without VLM access. The procedure executes an A* search over the task graph, interleaved with vertex refinements (but no edge refinements). The correctness and optimality guarantees both apply to this method.

DiRL (Jothimurugan et al. 2021). No refinements are conducted; this method relies fully on the structure of the input graph. The correctness guarantee is vacuous for this method, and the optimality guarantee applies.

5.2 Results

For all metrics, we report the mean over 4 or 5 experiment runs. Error bars denote the population standard deviation (computed using `np.std`).

Comparison to baselines. Our method successfully produces high-quality compositional controllers for the core task suite, consistently beating the main baseline method (VLM task graph refinement), as shown in Figure 4. The gap is most significant for the two rightmost tasks ("XX Mazes"). These are also the longest-horizon of the set, as they are compositional, consisting of two "X Maze" rooms.

Scaling to complex mazes and long-horizon tasks. We evaluate how our method's performance scales with task complexity by chaining together copies ("rooms") of the standard U Maze environment. Results are presented in Figure 5. Our method is unaffected by increasing task complexity, as the mean success rate remains constant as the task length increases. Notably, it also produces high-quality compositional

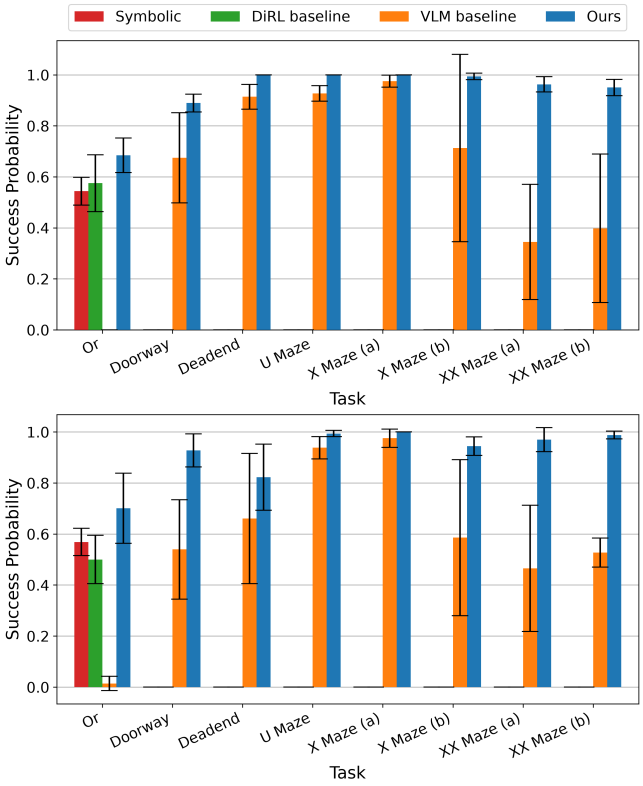


Figure 4: Our method (rightmost, blue) outperforms all 3 baselines on every task, as measured by mean success probability of the output compositional policy. These results are consistent for Gemini (upper) and OpenAI models (lower).

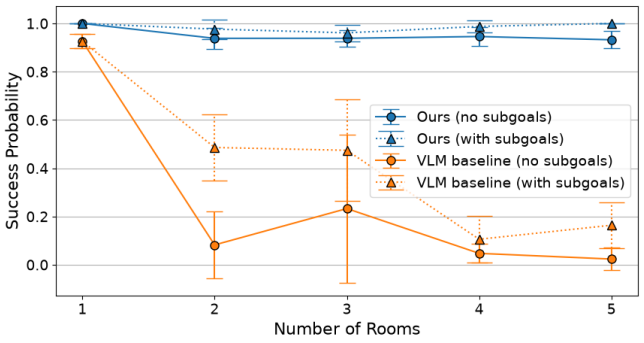


Figure 5: We compose the standard U Maze as a sequence of rooms. The input task graph may be a single edge ("no subgoals"), or a path with vertices positioned at the doorways between rooms ("with subgoals"). Each point averages 4-5 experiment runs using Gemini 3.5 Flash, and error bars denote the population standard deviation.

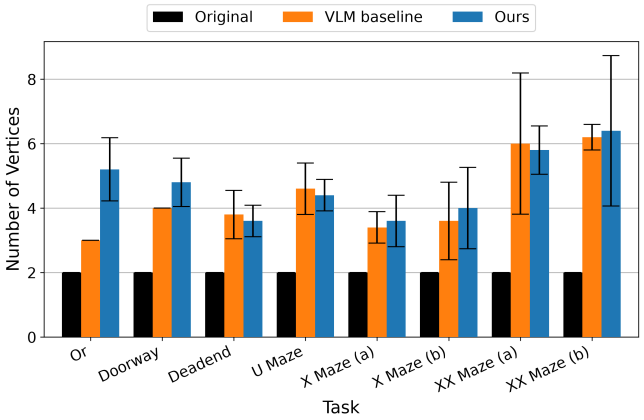


Figure 6: In each of these tasks, the input task graph is a single edge, representing a reach-avoid task. Using Gemini 3.5 Flash, our method produces task graphs whose verbosity is on a similar scale as the VLM baseline (but which induce consistently better compositional policies).

policies regardless of whether subgoals are provided. In contrast, the one-shot VLM baseline copes poorly with scaling task complexity. Surprisingly, the VLM baseline does not even benefit from being provided with denser initial specifications. This indicates that the VLM is unable to effectively make use of the information provided in a verbose input task graph. For tasks with multiple rooms, our method outperforms the baseline by $\geq 2\times$, regardless of the initial graph.

Drawbacks of the VLM baseline. Figure 5 demonstrates the weaknesses of the unscaffolded VLM as task complexity increases. When the number of rooms is still small (2-3), the VLM does benefit from receiving structural information in the form of a denser task graph, doubling or tripling the mean success rates of its output compositional policies. However, this behavior disappears when the task complexity increases further (4-5 rooms). These results indicate that without scaffolding, the VLM is unable to realize the benefit of providing an input task graph, even when this given information scales with the complexity of the task.

Solution complexity. Because our method produces a task graph as well as a compositional policy, the task graph should ideally be informative as well: for example, it should express the abstract solution strategy and may indicate bottlenecks in the solution path. Further, we might hope that the produced task graph is truly an abstract representation of the task, rather than simply storing a chain of waypoints; in other words, that it is concise enough to be interpretable. Figure 6 demonstrates that this is indeed the case. In most tasks, our method does not produce significantly more complex task graphs than the VLM baseline. However, in the two tasks where it is significantly more verbose ("Or" and "Doorway"), Figure 4 shows that its success probability is significantly higher, indicating that the extra verbosity is necessary for the quality of the compositional policy.

6 Conclusion

We present a procedure that produces interpretable compositional policies guaranteed to satisfy the input formal task specification. The method consistently matches or outperforms the unscaffolded VLM baseline, and generates high-quality solutions for long-horizon tasks where baselines fail.

References

Ambadkar, T.; Žikelić, Đ.; and Verma, A. 2025. Automating the Refinement of Reinforcement Learning Specifications. *arXiv preprint arXiv:2512.01047*.

Andreas, J.; Klein, D.; and Levine, S. 2017. Modular multitask reinforcement learning with policy sketches. In *International conference on machine learning*, 166–175. PMLR.

Cao, Y.; Zhao, H.; Cheng, Y.; Shu, T.; Chen, Y.; Liu, G.; Liang, G.; Zhao, J.; Yan, J.; and Li, Y. 2024. Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods. *IEEE Transactions on Neural Networks and Learning Systems*.

de Lazcano, R.; Andreas, K.; Tai, J. J.; Lee, S. R.; and Terry, J. 2024. Gymnasium Robotics.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2022. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73: 173–208.

Jackermeier, M.; and Abate, A. 2024. Deepltl: Learning to efficiently satisfy complex ltl specifications for multi-task rl. *arXiv preprint arXiv:2410.04631*.

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

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

Jothimurugan, K.; Bastani, O.; and Alur, R. 2021. Abstract value iteration for hierarchical reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, 1162–1170. PMLR.

Kim, J.; Park, S.; and Kim, G. 2021. Unsupervised skill discovery with bottleneck option learning. *arXiv preprint arXiv:2106.14305*.

Liang, W.; Wang, S.; Wang, H.-J.; Bastani, O.; Jayaraman, D.; and Ma, Y. J. 2024. Environment curriculum generation via large language models. In *8th Annual Conference on Robot Learning*.

Ma, Y. J.; Liang, W.; Wang, G.; Huang, D.-A.; Bastani, O.; Jayaraman, D.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*.

OpenAI. 2025. GPT-5.

Pateria, S.; Subagdja, B.; Tan, A.-h.; and Quek, C. 2021. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5): 1–35.

Ramesh, R.; Tomar, M.; and Ravindran, B. 2019. Successor options: An option discovery framework for reinforcement learning. *arXiv preprint arXiv:1905.05731*.

Rozier, K. Y. 2011. Linear temporal logic symbolic model checking. *Computer Science Review*, 5(2): 163–203.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shah, D.; Osiński, B.; Levine, S.; et al. 2023. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on robot learning*, 492–504. pmlr.

Stolle, M.; and Precup, D. 2002. Learning options in reinforcement learning. In *Abstraction, Reformulation, and Approximation: 5th International Symposium, SARA 2002 Kananaskis, Alberta, Canada August 2–4, 2002 Proceedings 5*, 212–223. Springer.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211.

Team, G.; Anil, R.; Borgeaud, S.; Alayrac, J.-B.; Yu, J.; Soricut, R.; Schalkwyk, J.; Dai, A. M.; Hauth, A.; Millican, K.; et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.

Towers, M.; Kwiatkowski, A.; Terry, J.; Balis, J. U.; De Cola, G.; Deleu, T.; Goulão, M.; Kallinteris, A.; Krimmel, M.; KG, A.; et al. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032*.

Vaezipoor, P.; Li, A. C.; Icarte, R. A. T.; and Mcilraith, S. A. 2021. Ltl2action: Generalizing ltl instructions for multi-task rl. In *International Conference on Machine Learning*, 10497–10508. PMLR.

Wang, L.; Ling, Y.; Yuan, Z.; Shridhar, M.; Bao, C.; Qin, Y.; Wang, B.; Xu, H.; and Wang, X. 2024a. GenSim: Generating Robotic Simulation Tasks via Large Language Models. In *The Twelfth International Conference on Learning Representations*.

Wang, Y.; Xian, Z.; Chen, F.; Wang, T.-H.; Wang, Y.; Fragkiadaki, K.; Erickson, Z.; Held, D.; and Gan, C. 2024b. RoboGen: Towards Unleashing Infinite Data for Automated Robot Learning via Generative Simulation. In *Forty-first International Conference on Machine Learning*.

Yu, W.; Gileadi, N.; Fu, C.; Kirmani, S.; Lee, K.-H.; Arenas, M. G.; Chiang, H.-T. L.; Erez, T.; Hasenclever, L.; Humplik, J.; et al. 2023. Language to Rewards for Robotic Skill Synthesis. In *Conference on Robot Learning*, 374–404. PMLR.

Zhang, H.; Wang, H.; Huang, X.; Chen, W.; and Kan, Z. 2024. Exploiting Hybrid Policy in Reinforcement Learning for Interpretable Temporal Logic Manipulation. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 13795–13800. IEEE.