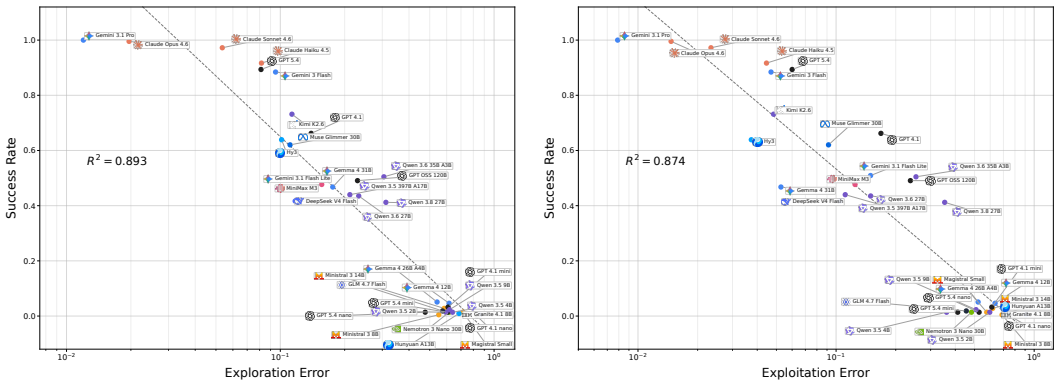
EXPLORATION AND EXPLOITATION ERRORS ARE MEASURABLE FOR LANGUAGE MODEL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language Model (LM) agents are increasingly used in complex open-ended decision-making tasks, from AI coding to physical AI, where they must both explore the problem space and exploit acquired knowledge effectively. However, it remains challenging to systematically distinguish exploration and exploitation using observed actions without access to the agent's internal policy, and to determine when either constitutes an error. To address this, we design partially observable 2D grid maps paired with an unknown symbolic task directed acyclic graph (DAG), abstracting the structure of practical agentic AI scenarios. We then propose a metric that detects unproductive traversals using redundancy criteria grounded in classical graph exploration bounds, while permitting necessary backtracking. We attribute each error to exploration or exploitation based on whether the current state admits productive exploration, exploitation, or both, making the metric policy-agnostic. Using programmatically generated maps and task DAGs with controllable difficulty, we evaluate 35 frontier LM agents and find that even state-of-the-art models struggle to reach the goal within the step budget. Log exploration and exploitation errors exhibit strong linear relationships with task success ($R^2 = 0.893$ and 0.874, respectively), providing empirical evidence that the metric captures behavior relevant to task success. Beyond success rate, the metric reveals distinct exploration and exploitation behaviors among models with similar success rates. We further find that a compact memory harness achieves higher task success than a reasoning baseline that retains the model's own reasoning traces. Finally, we extend our metric to an iconic video game, Minecraft Clone, beyond our synthetic environment.



(a) Success Rate vs. Exploration Error

(b) Success Rate vs. Exploitation Error

Figure 1: Overall relationship between success rate and exploration/exploitation errors across various LMs. In Figures 1a and 1b, log exploration and exploitation errors show a strong negative linear relationship with success rate ($R^2 = 0.893$ and 0.874, respectively). This indicates that our exploration and exploitation error metric captures behavior relevant to task success. Full results, including the exploration and exploitation errors, and average budget used is reported in Appendix C.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Language Model (LM) agents play a crucial role in many open-ended decision-making tasks, including AI coding (Jimenez et al., 2024; Jain et al., 2025; Merrill et al., 2026), workflow automation (Anthropic, 2024; Li et al., 2024; Lopopolo, 2026; Lee et al., 2026), and physical AI (Huang et al., 2022; Wang et al., 2023; Gubernatorov et al., 2025). By nature, these tasks require agents to explore unseen regions of the problem space and exploit acquired knowledge effectively (Harris & Slivkins, 2025; Kim et al., 2026; Jeong et al., 2026). Despite the growing adoption and strong performance of LM agents on various complex tasks, a systematic framework to assess and quantify how well these agents explore and exploit in practice does not yet exist.

In classical reinforcement learning (Sutton & Barto, 2018), exploration and exploitation are defined with respect to the agent's internal policy or value function. For LM agents, however, it is not straightforward because we typically only have access to the agent's observed actions. As a result, distinguishing and quantifying exploration and exploitation and identifying when either constitutes an error from behavior alone, without assuming a fixed strategy or access to the agent's internal policy, remains an open problem.

To address this, we introduce a policy-agnostic metric for quantifying exploration and exploitation errors from action trajectories alone. We instantiate tasks as partially observable 2D grid maps paired with unknown task directed acyclic graphs (DAGs) to enable systematic evaluation of exploration and exploitation from observed actions. This formulation abstracts the structure common to practical agentic tasks: navigating a partially observed space while achieving tasks with complex dependencies. A key design choice is replacing all semantic information in the task DAGs with symbolic representations. The agent must reason purely from the observed environment, effectively preventing pretrained semantic priors from confounding the evaluation.

Our error metric flags actions which no reasonable strategy would produce. Rather than specifying an optimal policy and penalizing deviation, we characterize the map state at each timestep and detect structurally redundant behavior within segments of the agent trajectory where no progress towards the task is being made. Our metric is inspired by classical graph-theoretic notions of redundancy and reuse (Whitney, 1932; Tarjan, 1972; Edmonds & Johnson, 1973; Deng & Papadimitriou, 1999; Panaite & Pelc, 1999), and attributes each error to exploration, exploitation or both.

With the proposed metric, we evaluate 35 frontier LM agents across diverse map configurations. The maps can be generated to require more exploration or exploitation by varying node density, corridor width, and task DAG size. We find that even state-of-the-art models often struggle on our task, and that log exploration and exploitation errors strongly correlate with task success. Moreover, models with similar success rates exhibit distinct exploration and exploitation behaviors. We further ablate the effects of various configurations including prompt types, semantic information, and explicit agent harnesses. Notably, we find that both exploration and exploitation can be improved through minimal harness engineering, achieving higher success than a reasoning baseline. Finally, we extend our metric to the clone version of a well-known video game, Minecraft Clone (Ouyang et al., 2026).

Summarizing, our contributions are threefold:

- We introduce a policy-agnostic metric for quantifying exploration and exploitation errors in LM agents from action trajectories, and extend it to Minecraft Clone;
- We design partially observable grid-map environments paired with unknown symbolic task DAGs, enabling systematic evaluation under varying exploration and exploitation demands;
- We evaluate 35 frontier LM agents, find that our errors strongly correlate with task success, identify distinct failure modes, and show that minimal harness engineering reduces both errors.

2 RELATED WORK

Language Model Agents. Language Model (LM) agents interact with an external environment in a multi-turn setting to solve complex problems (Yao et al., 2023; Shinn et al., 2023). To tackle a complex task, LM agents must explore the external environment to acquire new information and exploit it to achieve their goals. This has motivated a growing body of benchmarks that evaluate agent behavior across embodied interaction, software tasks, and tool-use settings (Shridhar et al.,

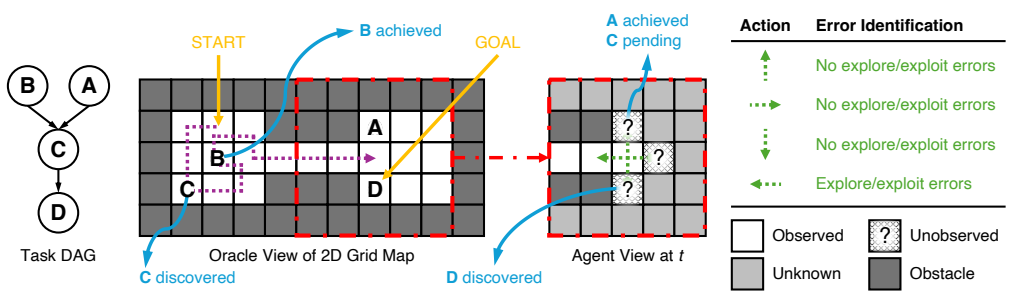


Figure 2: Illustration of the LM agent traversing the 2D grid map to achieve the goal given by the task DAG. In each timestep, the environment returns the set of admissible moves and the information about the task node, if discovered. The LM agent must retain relevant information about the map geometry, the task DAG structure and the states of the discovered nodes to achieve the goal node.

2021; Jimenez et al., 2024; Trivedi et al., 2024; Merrill et al., 2026), while partially observable grid world environments have been adopted for more controlled and systematic analysis of the LM agents (Chevalier-Boisvert et al., 2019; 2023). More recently, Zhang et al. (2026) assessed whether LM agents can construct and exploit spatial beliefs through exploration.

However, existing environments generally (i) rely on semantic information which confounds pre-trained knowledge with in-environment reasoning, (ii) lack systematic control over task dependency structure, and (iii) do not separate and quantify exploration and exploitation errors from the LM agent trajectories. In this work, we isolate LM-agent reasoning through symbolic task DAGs, use controllable task DAG generation to vary dependency structure, and propose a metric that quantifies exploration and exploitation errors.

Evaluation Metrics for LM Agents' Behavior. Current agent evaluation methods predominantly rely on task success rates only (Shridhar et al., 2021; Merrill et al., 2026). While some recent studies propose more fine-grained metrics such as stepwise alignment with expected tool calls or progress compared to a reference trajectory (Chen et al., 2024; Ma et al., 2024), these methods implicitly assume access to annotated reference trajectories and thus a fixed optimal strategy. Furthermore, these works do not distinguish whether errors stem from exploration or exploitation. In contrast, our exploration and exploitation error metric, grounded in classical graph theory (Whitney, 1932; Tarjan, 1972; Deng & Papadimitriou, 1999; Panaite & Pelc, 1999), does not rely on a particular reference trajectory. Instead, our proposed metric structurally analyzes the map state at each timestep to detect actions that no reasonable strategy would produce, and attributes each error to exploration, exploitation, or both based on the map state. It is therefore policy-agnostic and can be used to evaluate diverse LM agents that may internally follow different strategies. On the other hand, the behavior of LLM reasoning has been discussed in the work by Zeng et al. (2026), devising their own metrics on the tree representation of reasoning traces.

3 Task Formulation

In this section, we formally define the design of our framework and show that the specific design choices in our framework allow us to distinguish and quantify exploration and exploitation. At a high level, the LM agent must traverse the 2D grid map to discover relevant task nodes, use its accumulated knowledge to satisfy their prerequisites, and ultimately achieve the goal node. For a more mathematical definition, refer to Appendix B.

2D Grid Map. We define our partially observable map $\mathcal{M}$ as the set of traversable[1] cells in a 2D grid, analogous to real-world or video game settings where movement gradually reveals local spatial information and any task-relevant entities present. More formally, $\mathcal{M} \subset \mathbb{N}^2$ where each element in $\mathcal{M}$ is a coordinate of a cell.[2] For example, $\mathcal{M} = \{0, 1, 2\}^2$ denotes a fully traversable 3×3 2D grid. The map allows `up`, `down`, `left`, `right` as movements.

[1]For example, obstacles are not included in $\mathcal{M}$.

[2]We assume that $0 \in \mathbb{N}$.

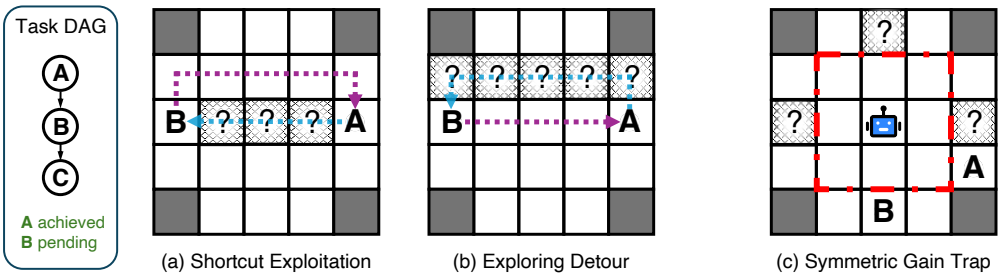


Figure 3: Three illustrative examples demonstrating edge cases. In all three maps, node A is achieved and node B is pending, so the agent can traverse back to B, or explore to discover node C. The purple dashed line depicts the path the LM agent has taken, and the blue transparent line denotes a potentially more optimal path. In Panel (a), the LM agent can infer the shortest path to node A and traverse unseen nodes as part of an exploitation strategy. In contrast, in Panel (b), the agent has already taken the shortest path from node B to node A. Hence, it may be more optimal to make a short detour and observe 5 unseen cells at the cost of a short detour. Panel (c) illustrates a case where target cells $\mathcal{T}(t)$ are placed symmetrically. Here gain will always be 1 for all four actions. A metric solely relying on gain cannot penalize the LM agent wandering indefinitely in the red dotted region.

Each time the agent visits a cell, the environment provides a subset of `up`, `down`, `left`, `right` as admissible moves, which provide information about the neighboring cells (Figure 2). We say a cell c is **observed** if the agent has visited the cell c at some previous timestep. The neighboring cells that have not been previously observed are denoted **unobserved**. Critically, information about the task DAG is only revealed when the cell is observed. The agent must traverse to unobserved cells to discover if task nodes are present in those cells. If no task node is present, the cell is simply marked as observed. Finally, cells that are neither observed nor unobserved are **unknown**; see Equation (2) for formal definitions.

Task DAG. Complex real-world tasks can be decomposed into sub-tasks with precedence constraints, and are naturally modeled as DAGs, where sub-tasks are nodes and edges define the precedence relations between the nodes. For example, cooking tomato pasta with cheese requires finding tomato sauce, pasta, and cheese, and mixing them in the right order (see Figure 9). Without loss of generality, we assume a unique goal node in our setting.[3]

Denote $l(\cdot)$ as an injective function between the task node and its location. We say a node u is **visited** if the agent is currently located at $l(u)$. If the agent has found node u at some previous timestep, a node u has been **seen**. With this, we define the three states each node has: $\{\text{undiscovered}, \text{discovered}, \text{achieved}\}$. A node is **undiscovered** if it has not been seen. A node is **discovered** if it has been seen but the preconditions are not met. A node is **achieved** only if the agent has visited the node after satisfying the preconditions; see Equation (3). Finally, each node has two types of preconditions: $\{\texttt{AND}, \texttt{OR}\}$ where `AND` indicates that all of its parent nodes must be achieved, and `OR` indicates that only one of its parent nodes needs to be achieved in order to achieve the current node (see Equation (4)).

Upon visiting a cell in the 2D grid map, the environment provides the set of admissible movements. If a node is discovered, the environment provides information about its parent and child nodes. The locations of the revealed nodes remain unknown and must be discovered by the agent. The task is complete when the goal node is achieved.

Agent Harness. In each interaction with the environment, the following information must be tracked by the LM agent to effectively solve the task.

- Map information: list of observed and unobserved cells and obstacles, if present.
- Task information: list of achieved and discovered task nodes, and their preconditions.

[3]If there are multiple goal nodes, we can append a dummy sink node.

Table 1: Four cases determining the type of required action at timestep t.

Case	Condition	Target set $\mathcal{T}(t)$	Required action
1	$\mathcal{P}(t) = \emptyset$	$\mathcal{U}(t)$	Exploration
2	$g \in \mathcal{P}(t)$	$\{l(g)\}$	Exploitation
3	$\mathcal{P}(t) \neq \emptyset,\ g \notin \mathcal{P}(t),\ \mathcal{U}(t) = \emptyset$	$\{l(u) : u \in \mathcal{P}(t)\}$	Exploitation
4	$\mathcal{P}(t) \neq \emptyset,\ g \notin \mathcal{P}(t),\ \mathcal{U}(t) \neq \emptyset$	$\mathcal{U}(t) \cup \{l(u) : u \in \mathcal{P}(t)\}$	Either

In practice, LM agents operating over long horizons benefit from structured summaries of accumulated state, rather than relying solely on raw context history (Anthropic, 2025). Hence, we can provide a subset of the available information and define this as the **agent harness**. Following the philosophy of ReAct (Yao et al., 2023), we provide minimal information to the agent by default and assume that a skilled agent would maintain relevant context history and make rational decisions at each timestep. In Section F, we ablate explicitly providing the full harness to the agent. Providing the harness explicitly is analogous to equipping the agent with an anchored map (Dessmark & Pelc, 2004) or an advice string (Fraigniaud et al., 2008; Dobrev et al., 2012), where structured information about the environment is made directly available rather than requiring reconstruction from context.

4 Measuring Exploration and Exploitation Errors

In this section, we formally define our metric for exploration and exploitation errors.

We define **pending tasks** $\mathcal{P}(t)$ which denotes the nodes in the task DAG with prerequisites satisfied and locations known, *making them ready to be achieved* (see Equation (5)). This reflects many real-world tasks where, once the prerequisite conditions are satisfied, progress requires traversing back to the relevant location to act. Intuitively, the LM agent can exploit its knowledge to achieve the pending tasks. The exploration counterpart to $\mathcal{P}(t)$ is the set of unobserved cells $\mathcal{U}(t)$, which the agent must traverse to in order to discover new information. We refer to the set of productive destinations as the **target set** $\mathcal{T}(t)$, which varies with the agent's state and determines the required action at timestep t (Table 1).

That is, we assume that the agent must explore when $\mathcal{U}(t) \neq \emptyset$ and exploit when $\mathcal{P}(t) \neq \emptyset$, with the exception when the goal task is pending, where we only allow exploitation.

We aim to identify actions that no reasonable strategy would produce as errors. We say that an action is a **gain** (Equation (6)) if it steps into a target cell or reduces the minimum distance to at least one of the target cells (Gain$(t \to t+1) = 1$), and define the action as an error otherwise (Gain$(t \to t+1) = 0$). However, as shown in Figure 3c, gain alone is insufficient to classify errors: when targets exist symmetrically, the agent can oscillate indefinitely.

To address this, we define a **no-progress trajectory** $\tau_{\text{np}}(t)$ as the sequence of actions since the most recent progress event, where a **progress event** is either achieving a pending task or entering an unobserved cell. $\tau_{\text{np}}(t) = \emptyset$ once progress is made. We track the history of nodes and edges the agent has traversed in each $\tau_{\text{np}}(t)$ and denote them $\mathcal{V}_{\text{np}}$ and $\mathcal{E}_{\text{np}}$ respectively, and compute the following three quantities:

- $c_t = |\mathcal{E}_{\text{np}}| - |\mathcal{V}_{\text{np}}| + 1$, the cyclomatic number of the current no-progress trajectory.
- $e_t = \sum_{e \in \mathcal{E}_{\text{np}}} \max\{m_{\text{np}}(e) - 2, 0\}$, where $m_{\text{np}}(e)$ is the traversal count of e in τ_{np}.
- $n_t = \sum_{v \in \mathcal{V}_{\text{np}}} \max\{m_{\text{np}}(v) - 2, 0\}$, where $m_{\text{np}}(v)$ is the visit count of v in τ_{np}.

Intuitively, c_t increases when the agent closes a new loop in explored territory (Whitney, 1932). e_t and n_t increase when the edge or node is reused beyond benign backtracking, respectively. More formally, the budget of 2 for edges is motivated by classical graph exploration, where optimal online exploration of an undirected graph traverses each undirected edge at most twice (Tarjan, 1972; Edmonds & Johnson, 1973; Panaite & Pelc, 1999; Deng & Papadimitriou, 1999). The budget of 2 for nodes is the node-level analog: the tightest constant that permits a single gateway revisit without penalty. Although we do not enforce optimal traversal, we assume that a competent agent should

avoid structurally redundant actions that yield no new information. Empirically, this formulation captures a wide range of failure modes, some of which are listed here in Appendix B.

Now we define the **stale score** $S_t = c_t + e_t + n_t$ and flag error when the stale score increases – i.e. $\mathbb{1}\{S_t > S_{t-1}\}$.[4] Combining the above, we define the error metric at timestep t:

$$\mathrm{err}(t) = \begin{cases} 0, & \text{if } p(t) \to p(t+1) \text{ is a progress event,} \\ 1, & \text{if Gain}(t \to t+1) = 0, \\ 0, & \text{if } |\mathcal{T}(t)| = 1 \wedge \text{Gain}(t \to t+1) = 1, \\ \mathbb{1}\{S_t > S_{t-1}\}, & \text{if } |\mathcal{T}(t)| > 1 \wedge \text{Gain}(t \to t+1) = 1. \end{cases} \tag{1}$$

The stale score is only required when more than one cell exist in the target set $\mathcal{T}(t)$ (Figure 3c). Case 2 implies $|\mathcal{T}(t)| = 1$. When $\mathrm{err}(t) = 1$, we attribute the error to exploration, exploitation or both, according to the required action in Table 1: Case 1 increments exploration error, Case 2 and Case 3 increment exploitation error, and Case 4 increments both errors. For more detailed discussion on what each of the four cases are, see Appendix B.

5 Experimental Setup

We cover detailed settings for our experiments. For full details on task generation and data statistics, refer to Appendices D.1 and D.2, respectively.

Task Generation. We first generate task DAGs, then lay them out on 2D grid maps. Then, we place the nodes on the 2D grid, by varying the following parameters: (i) the number of task nodes; (ii) maximum number of nodes at each depth; (iii) the node density over grid cells. We place obstacle cells to control the width of the path from one node to another. This implicitly controls the degree of exploration and exploitation needed.

Models. We evaluate 35 LMs. The full list is in the Appendix C.

Prompts. By default, we follow ReAct (Yao et al., 2023) framework and consider four prompt variants: **base**, **exploration**, **exploitation**, and **balance**. All four variants share an identical environment description and action format; they differ only in a single strategy sentence that instructs the agent to prioritize exploration, exploitation, or balance the two. The base prompt provides no strategic guidance, leaving the exploration–exploitation tradeoff entirely to the model's internal context and reasoning capabilities. Full prompt templates and example interactions are provided in Appendix D.3.

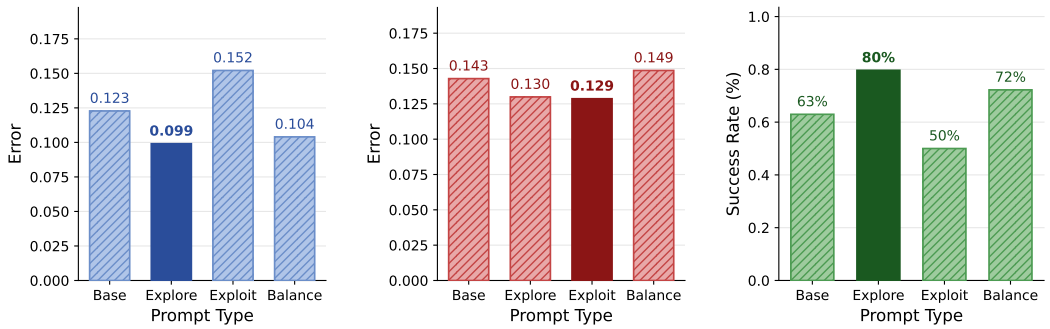
Evaluation. We report **success rate**, **exploration error**, and **exploitation error**. Success rate measures the fraction of episodes the agent achieves the goal within the step budget. Error metrics are normalized by the number of timesteps in which each action type is required: exploration error over Cases 1 and 4, and exploitation error over Cases 2, 3, and 4 (Table 1).

6 Experimental Results

Figure 1 illustrates our main results comparing the success rate versus exploration or exploitation error with various frontier LM agents. Stronger reasoning models consistently achieve higher performance, with the best models reaching up to 100% success rate. These results exhibit strong negative relationships between success rate and both exploration and exploitation errors, with exploration error showing a marginally stronger association. This is consistent with the structure of our task: agents must first discover task-relevant nodes through exploration and subsequently exploit acquired knowledge to satisfy task dependencies and reach the goal. Persistent failures in either process can therefore limit task completion, although exploration failures may act as an earlier bottleneck by preventing relevant states from being discovered. Therefore, we argue the following:

> ***Finding 1***: Low exploration and exploitation error rates are strong predictors of success.

[4] For detailed description about this metric, please refer to Appendix B.



(a) Exploration Error (b) Exploitation Error (c) Success Rate

Figure 4: Effect of prompts on exploration and exploitation error, and success rate for GPT-4.1. The highlighted bar in each panel indicates lowest error or highest success rate. The exploration-focused prompt achieves the lowest exploration error and the highest overall success rate, while the exploitation-focused prompt achieves the lowest exploitation error.

Table 2: Effect of harness engineering on success rates, exploration and exploitation errors, and average budget used. Harness engineering provides a structured memory summary and the model outputs actions without reasoning (Appendix F). Compared to reasoning, it achieves higher success rates, lower exploration errors, and lower total budget usage for both models.

Model	Method	Success (%) ↑	Exploration Error ↓	Exploitation Error ↓	Avg. Budget Used (Total) (%) ↓	Avg. Budget Used on Success (%) ↓
Gemini 3.1 Flash Lite	Reasoning	50.0	0.190	0.193	77.1	**54.2**
	Harness Engineering	**76.9**	**0.144**	**0.083**	**64.9**	54.3
GPT-4.1	Reasoning	66.7	0.159	**0.227**	72.6	58.9
	Harness Engineering	**72.2**	**0.121**	0.293	**69.2**	**57.4**

We analyze the impact of providing prompts emphasizing exploration or exploitation by testing four prompt variants: base, exploration-focused, exploitation-focused, and balanced prompts. As depicted in Figure 4, exploration- and exploitation-focused prompts reduce exploration and exploitation errors, respectively. The highest success rate is achieved with exploration-focused prompts, consistent with the results in Figure 1 and Finding 1. Hence:

> ***Finding 2***: Exploration-focused or exploitation-focused prompts reduce exploration and exploitation errors, respectively.

As denoted in Section 3, one can design an explicit memory by acquiring relevant information the environment returns, which can be viewed as an external memory management system in the context of harness engineering (see Appendix F for an example), instead of relying on the LM agent's internal context and reasoning. We observe that providing structured harness to the LM agents, even without reasoning, improves success rates, exploration errors, and total budget usage compared to reasoning, except for the exploitation error of GPT-4.1 (Table 2).

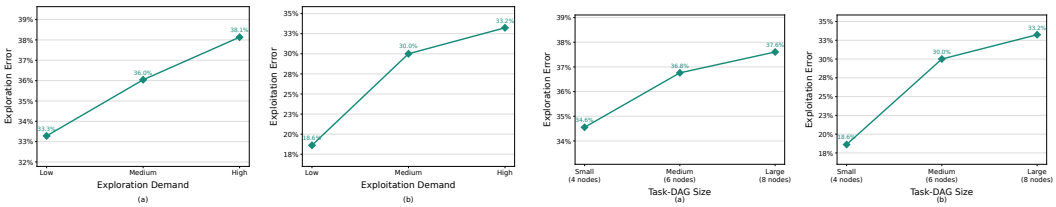
> ***Finding 3***: Harness engineering without reasoning outperforms reasoning by utilizing historical outcomes.

In our main experiments, we removed semantic information in our task DAG generation to isolate LM's semantic priors. In Table 3, we reintroduce semantic information by evaluating agents on simple pasta-cooking tasks (see Figure 9 for an example; experimental details are provided in Appendix G). For both models, semantic information substantially reduces exploitation error ($0.405 \rightarrow 0.061$ and $0.456 \rightarrow 0.219$), while exploration error remains similar. This suggests that semantic information helps LM agents infer the task dependencies, but does not reveal where the nodes are, so exploration is still required.

Table 3: Effect of semantic information for Gemini 3.1 Flash Lite and GPT-4.1. Semantic information reduces exploitation errors for both models, while exploration errors remain similar. Budget usage is the mean percentage of each map's hidden budget consumed, reported for successful trajectories and all trajectories, respectively. Each condition pools 40 evaluations over four maps, five seeds, and the default and reasoning evaluation.

Model	Semantic	Success (%) ↑	Exploration Error ↓	Exploitation Error ↓	Avg. Budget Used (Success / Total) ↓
Gemini 3.1 Flash Lite	✗	0.0	0.305	0.405	N/A / 100.0%
	✓	**25.0**	**0.291**	**0.061**	**80.1% / 95.0%**
GPT-4.1	✗	25.0	**0.275**	0.456	74.0% / 93.5%
	✓	**27.5**	0.284	**0.219**	**68.0% / 91.2%**

Finding 4: Reintroducing semantic information reduces exploitation errors, but not exploration errors.



(a) By Exploration Demand (by map design) (b) By Task DAG Size

Figure 5: Exploration and exploitation errors averaged over 35 models, with varying (a) exploration demand, controlled by map design, and (b) task DAG size. Each model is evaluated on 216 episodes. Errors increase monotonically with both exploration demand and task DAG size.

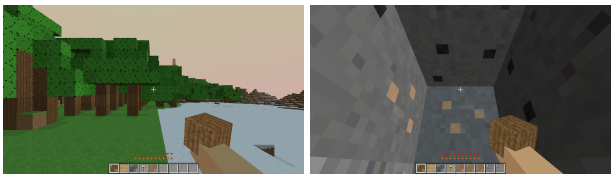
Figure 5 shows how exploration and exploitation errors vary with map-level demand and task DAG size. Both errors increase monotonically with their corresponding demand levels: exploration error rises as exploration demand increases, while exploitation error rises with exploitation demand. Similarly, increasing the task DAG size increases both exploration and exploitation errors. These results indicate that the proposed metrics respond systematically to controlled increases in the exploration and exploitation requirements of the task.

Finding 5: Exploration and exploitation errors increase systematically with their corresponding task demands and with task DAG size.

An interesting result is shown in Section E. While Hy3 and GPT-OSS-120B both achieve success in the same environment, the two models demonstrate different quantitative behaviors; one shows zero exploration error, while the other shows zero exploitation error. This example implies that the success rate is not enough to represent agents' behavior, and our error metric can convey more informative understanding of the agents.

7 Discussion and Limitations

On the use of symbolic abstraction. Our environment is intentionally not a full reproduction of real-world scenarios. As noted in Section 1, we remove semantic information and use symbolic task DAGs to isolate the raw exploration and exploitation capabilities of LM agents (the exact symbolic node generation process is detailed in Appendix D.1). However, a key limitation is that many real-world tasks will not appear in such a fully semantics-free form; in practice, agents are often expected to use semantic information and pretrained knowledge rather than operate in environments deliberately detached from them. This can be viewed as a limitation if the goal is to measure end-to-end real-world utility, since many practical applications allow and even require agents to leverage domain knowledge and priors to solve complex tasks more effectively. Indeed, our semantic reintroduction experiments show that such information can substantially affect agent behavior (Finding



(a) Observation Examples

Model	Memory Manage	Success Rate (%)	Exploration Error	Exploitation Error
Gemini 3.1 Flash Lite	✗	0.0	0.948	0.000
	✓	88.9	0.807	0.000
GPT-4.1	✗	11.1	0.877	N/A
	✓	77.8	0.740	0.556

(b) Results

Figure 6: Observation examples and results with Gemini 3.1 Flash Lite and GPT-4.1 on Minecraft Clone. Memory Manage indicates inclusion of up to two prior screenshots.

4). At the same time, this is precisely why the symbolic abstraction is useful: it lets us test whether an agent can genuinely explore, maintain relevant memory and state information, and act on newly actionable knowledge when these behaviors must be inferred from interaction history alone rather than from semantic shortcuts. In this sense, our task is best viewed as a controllable test suite to measure raw exploration and exploitation capabilities that complement evaluations in more realistic semantic settings.

Per-model variance and the error metric. Another limitation is that our exploration and exploitation errors reported in the tables are inherently trajectory-dependent, since the per-case normalization depends on how many timesteps fall into each case, which itself is shaped by the agent's chosen path. Importantly, each LM agent shows different success rates and the average number of steps for successful trajectories which will directly influence the normalization described in Section 5. As a result, the normalized metric values should be viewed primarily as behavioral summaries rather than as complete standalone measures that can be used to compare overall agent qualities. However, this does not weaken the validity of the metric itself: for any trajectory and map state, our error function still provides a principled and consistent classification of whether an action constitutes an exploration or exploitation error. We therefore view the proposed metric as a complementary metric to success rate, offering a more fine-grained analysis of LM agent behavior. More broadly, we believe this framework provides a foundation that can be extended to more realistic settings by incorporating semantic information, richer environment structures beyond 2D grid maps, and more complicated task dependencies.

Extension to video games. Figure 6 extends our evaluation beyond symbolic grid maps by adapting the proposed metrics to Minecraft Clone. This experiment illustrates how our framework can characterize agent behavior in a visual 3D environment from recorded interactions, without accessing the agent's internal policy. The error decomposition reveals a distinction that task success alone cannot explain: without memory management with prior screenshots, Gemini 3.1 Flash Lite achieves zero success despite zero measured exploitation error, while its exploration error of 0.948 is consistent with a discovery bottleneck. The memory management reduces exploration error from 0.948 to 0.807 for Gemini and from 0.877 to 0.740 for GPT-4.1, alongside success-rate increases from 0.0% to 88.9% and from 11.1% to 77.8%, respectively. The decomposition also identifies remaining exploitation errors for GPT-4.1 with the memory management, whose exploitation error is 0.556 despite its improved success rate.

8 Conclusion

In this work, we introduce a policy-agnostic metric to quantify exploration and exploitation errors in LM agents from action trajectories. Using partially observable grid-map environments with symbolic task DAGs, we evaluate 35 frontier LM agents and show that low exploration and exploitation errors are strongly associated with success, while LM agents with similar success rates still exhibit qualitatively different behaviors. Our experiments further show that prompt design, harness engineering, and semantic information substantially affect how LM agents balance exploration and exploitation. Finally, our metric extends to Minecraft Clone, demonstrating its applicability beyond our synthetic environment. Overall, our results provide a foundation for quantifying exploration and exploitation errors in LM agents beyond success rate, offering a more informative lens for evaluating and improving LM agents in complex open-ended tasks.

AI USE STATEMENT

During the preparation of this work, we used LLMs in the following:

- Writing assistance: An LLM was used for grammar correction, fluency checking, and refinement of author-written text;
- Code implementation: Codex and Claude Code were used to assist with implementing analysis scripts for processing experimental results and generating plots;
- Prompt revision: OpenAI ChatGPT, Codex, and Claude Code were used for revising prompts;
- Script execution: Codex and Claude Code were utilized to execute written scripts and CLI.

REPRODUCIBILITY STATEMENT

We provide the details necessary to reproduce our experiments throughout the appendix. Appendix D.1 describes the procedural task and map generation process and its parameters; subsequent sections report the experimental configurations, episode counts, random seeds, prompt templates, and agent harness details. We also provide the implementation for environment generation, evaluation, and computation of the proposed exploration and exploitation error metrics at https://anonymous.4open.science/r/symbolic-explore-exploit-colm-5AB5.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Anthropic. What is the model context protocol (MCP)? `https://modelcontextprotocol.io`, 2024.

Anthropic. Effective harnesses for long-running agents. `https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents`, 2025.

Anthropic. Claude 4.5 haiku model card. `https://www-cdn.anthropic.com/7aad69bf12627d42234e01ee7c36305dc2f6a970.pdf`, 2025.

Anthropic. Claude 4.6 opus model card. `https://www-cdn.anthropic.com/14e4fb01875d2a69f646fa5e574dea2b1c0ff7b5.pdf`, 2026a.

Anthropic. Claude 4.6 sonnet model card. `https://www-cdn.anthropic.com/bbd8ef16d70b7a1665f14f306ee88b53f686aa75.pdf`, 2026b.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 2002.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, et al. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2024.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and J K Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

DeepSeek-AI. DeepSeek-V4-Flash: Model card. `https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash`, 2026. Accessed: 2026-09-26.

Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.

Anders Dessmark and Andrzej Pelc. Optimal graph exploration without good maps. *Theoretical Computer Science*, 326(1):343–362, 2004.

Wenxuan Ding, Nicholas Tomlin, and Greg Durrett. Calibrate-Then-Act: Cost-aware exploration in LLM agents. *arXiv preprint arXiv:2602.16699*, 2026.

Stefan Dobrev, Rastislav Královič, and Euripides Markou. Online graph exploration with advice. In *Proceedings of the International Conference on Structural Information and Communication Complexity*, 2012.

Jack Edmonds and Ellis Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5:88–124, 12 1973.

Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Tree exploration with advice. *Information and Computation*, 206(11):1276–1287, 2008.

Google. Gemma 4 12B IT: Model card. `https://huggingface.co/google/gemma-4-12B-it`, 2026a. Accessed: 2026-09-26.

Google. Gemma 4 26B A4B IT: Model card. `https://huggingface.co/google/gemma-4-26B-A4B-it`, 2026b. Accessed: 2026-09-26.

Google. Gemma 4 31B IT: Model card. `https://huggingface.co/google/gemma-4-31B-it`, 2026c. Accessed: 2026-09-26.

Google DeepMind. Gemini 3.1 flash lite model card. `https://deepmind.google/models/model-cards/gemini-3-1-flash-lite/`, 2025a.

Google DeepMind. Gemini 3 flash model card. `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf`, 2025b.

Google DeepMind. Gemini 3.1 pro model card. `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-1-Pro-Model-Card.pdf`, 2026.

Konstantin Gubernatorov, Artem Voronov, Roman Voronov, Sergei Pasynkov, Stepan Perminov, Ziang Guo, and Dzmitry Tsetserukou. Anywherevla: Language-conditioned exploration and mobile manipulation. *arXiv preprint arXiv:2509.21006*, 2025.

Keegan Harris and Aleksandrs Slivkins. Should you use your large language model to explore or exploit? *arXiv preprint arXiv:2502.00225*, 2025.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

IBM. Granite 4.1 8B: Model card. `https://huggingface.co/ibm-granite/granite-4.1-8b`, 2026. Accessed: 2026-09-26.

Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? scaling LLM inference-time compute with adaptive branching tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Jongwon Jeong, Jungtaek Kim, and Kangwook Lee. TAPE: Tool-guided adaptive planning and constrained execution in language model agents. In *ICLR Workshop on Agentic AI in the Wild: From Hallucinations to Reliable Autonomy (AAIW)*, 2026.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world Github issues? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Jungtaek Kim, Thomas Zeng, Ziqian Lin, Minjae Lee, Chungpa Lee, Jy-yong Sohn, Hyung Il Koo, and Kangwook Lee. Transformers in the Dark: Navigating unknown search spaces via bandit feedback. *Transactions on Machine Learning Research*, 2026.

Akshay Krishnamurthy, Keegan Harris, Dylan J Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context? *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond A*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*, 2024.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. AutoFlow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821*, 2024.

Ryan Lopopolo. Harness engineering: leveraging codex in an agent-first world. https://openai.com/index/harness-engineering/, 2026.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn LLM agents. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command kine interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

Meta Superintelligence Lab. Muse Glimmer-30B: Model card. https://huggingface.co/meta-models/Muse-Glimmer-30B, 2026. Accessed: 2026-09-26.

MiniMax. MiniMax-M3: Model card. https://huggingface.co/MiniMaxAI/MiniMax-M3, 2026. Accessed: 2026-09-26.

Mistral AI. Magistral Small 2509: Model card. https://huggingface.co/mistralai/Magistral-Small-2509, 2025a. Accessed: 2026-09-26.

Mistral AI. Ministral 3 14B Instruct 2512: Model card. https://huggingface.co/mistralai/Ministral-3-14B-Instruct-2512, 2025b. Accessed: 2026-09-26.

Mistral AI. Ministral 3 14B Reasoning 2512: Model card. https://huggingface.co/mistralai/Ministral-3-14B-Reasoning-2512, 2025c. Accessed: 2026-09-26.

Mistral AI. Ministral 3 8B Instruct 2512: Model card. https://huggingface.co/mistralai/Ministral-3-8B-Instruct-2512, 2025d. Accessed: 2026-09-26.

Mistral AI. Ministral 3 8B Reasoning 2512: Model card. https://huggingface.co/mistralai/Ministral-3-8B-Reasoning-2512, 2025e. Accessed: 2026-09-26.

Moonshot AI. Kimi K2.6: Model card. https://huggingface.co/moonshotai/Kimi-K2.6, 2026. Accessed: 2026-09-26.

NVIDIA Corporation. NVIDIA Nemotron 3 Nano 30B A3B BF16: Model card. https://huggingface.co/nvidia/NVIDIA-Nemotron-3-Nano-30B-A3B-BF16, 2025. Accessed: 2026-09-26.

OpenAI. GPT-4.1. https://openai.com/index/gpt-4-1/, 2025.

OpenAI. GPT-5.4 thinking system card. https://deploymentsafety.openai.com/gpt-5-4-thinking/gpt-5-4-thinking.pdf, 2026. Published March 5, 2026.

Mingyu Ouyang, Siyuan Hu, Kevin Qinghong Lin, Hwee Tou Ng, and Mike Zheng Shou. Gameworld: Towards standardized and verifiable evaluation of multimodal game agents. *arXiv preprint arXiv:2604.07429*, 2026.

Petrişor Panaite and Andrzej Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33(2):281–295, 1999.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

Qwen Team. Qwen3.5-2B: Model card. https://huggingface.co/Qwen/Qwen3.5-2B, 2026a. Accessed: 2026-09-26.

Qwen Team. Qwen3.5-397B-A17B: Model card. https://huggingface.co/Qwen/Qwen3.5-397B-A17B, 2026b. Accessed: 2026-09-26.

Qwen Team. Qwen3.5-4B: Model card. https://huggingface.co/Qwen/Qwen3.5-4B, 2026c. Accessed: 2026-09-26.

Qwen Team. Qwen3.5-9B: Model card. https://huggingface.co/Qwen/Qwen3.5-9B, 2026d. Accessed: 2026-09-26.

Qwen Team. Qwen3.6-27B: Model card. https://huggingface.co/Qwen/Qwen3.6-27B, 2026e. Accessed: 2026-09-26.

Qwen Team. Qwen3.6-35B-A3B: Model card. https://huggingface.co/Qwen/Qwen3.6-35B-A3B, 2026f. Accessed: 2026-09-26.

Qwen Team. Qwen3.8-27B: Model card. https://huggingface.co/Qwen/Qwen3.8-27B, 2026g. Accessed: 2026-09-26.

Alessio Russo, Ryan Welch, and Aldo Pacchiano. In-context learning for pure exploration. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Andrew Szot, Michael Kirchhof, Omar Attia, and Alexander Toshev. Expanding LLM agent boundaries with strategy-guided exploration. *arXiv preprint arXiv:2603.02045*, 2026.

Hao Tang, Keya Hu, Jin P Zhou, Sicheng Zhong, Wei-Long Zheng, Xujie Si, and Kevin Ellis. Code repair with LLMs gives an exploration-exploitation tradeoff. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2): 146–160, 1972.

Tencent. Hunyuan-A13B-Instruct: Model card. `https://huggingface.co/tencent/Hunyuan-A13B-Instruct`, 2025. Accessed: 2026-09-26.

Tencent. Hy3: Model card. `https://huggingface.co/tencent/Hy3`, 2026. Accessed: 2026-09-26.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 1933.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Z.ai. GLM-4.7-Flash: Model card. `https://huggingface.co/zai-org/GLM-4.7-Flash`, 2026. Accessed: 2026-09-26.

Yuchen Zeng, Shuibai Zhang, Wonjun Kang, Shutong Wu, Lynnix Zou, Ying Fan, Heeju Kim, Ziqian Lin, Jungtaek Kim, Hyung Il Koo, Dimitris Papailiopoulos, and Kangwook Lee. ReJump: A tree-jump representation for analyzing and improving LLM reasoning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026.

Pingyue Zhang, Zihan Huang, Yue Wang, Jieyu Zhang, Letian Xue, Zihan Wang, Qineng Wang, Keshigeyan Chandrasegaran, Ruohan Zhang, Yejin Choi, et al. Theory of space: Can foundation models construct spatial beliefs through active exploration? *Proceedings of the International Conference on Learning Representations (ICLR)*, 2026.

APPENDICES

Here, we provide additional details that were left out due to limited space.

A ADDITIONAL RELATED WORK

Exploration and Exploitation. The exploration–exploitation tradeoff has been extensively studied in reinforcement learning (RL) (Thompson, 1933; Auer et al., 2002; Bellemare et al., 2016; Pathak et al., 2017), and has recently gained attention in the context of LM agents (Tang et al., 2024; Inoue et al., 2025). Several works have shown that LMs do not explore efficiently, and have proposed methods to enhance exploration efficiency through in-context learning (Krishnamurthy et al., 2024; Russo et al., 2026), prompting (Ding et al., 2026), supervised training (Kim et al., 2026), and RL training (Szot et al., 2026). Likewise, many works report the limited exploitation capabilities of LM agents and address them through training (Lehnert et al., 2024), or integrating external planners (Jeong et al., 2026). More recently, Harris & Slivkins (2025) evaluated LMs' exploration–exploitation tradeoff in bandit settings and found that frontier models underperform simple linear regression baselines.

Prior works have focused on improving exploration or exploitation, and analyzing their tradeoffs in fixed environments where actions are independent to the environments. However, in real-world tasks, prior actions often determine what the LM agent can observe and achieve. To the best of our knowledge, no existing framework *quantitatively separates and measures* exploration and exploitation errors in such environments. Our work addresses this gap through the proposed 2D grid-map environments with task DAGs and the proposed exploration and exploitation metric.

B DETAILS OF TASK FORMULATION AND EXPLORATION AND EXPLOITATION METRIC

In this section, we provide the detailed intuition, metric design rationale, and illustrative examples that were deferred from the main text due to limited space.

2D Grid Map. For a cell $c \in \mathcal{M}$, we denote $\text{adj}(c)$ as the set of neighboring cells of c - i.e. cells that can be traversed from c.

We use $p(t) \in \mathcal{M}$ to denote the position of the agent at timestep t. Each time the agent visits a cell $p(t)$, the environment provides a subset of `up, down, left, right` as admissible moves, which provide information about the neighboring cells of $p(t)$ – i.e. $\text{adj}(p(t))$.

We say a cell c is **observed** at timestep t if the agent has visited the cell c at some previous timestep, i.e. $\exists t' \leq t : p(t') = c$. Neighboring cells of $p(t)$ that have not been previously observed are defined as **unobserved**. Critically, information about the task DAG is only revealed when the cell is observed. The agent must traverse to unobserved cells to discover if task nodes are present in those cells. If no task node is present, the cell is simply marked as observed. Finally, cells that are neither observed nor unobserved states are defined as **unknown**. Summarizing, we define the state function for cell c at timestep t as such:

$$\text{state}(c,t) = \begin{cases} \text{observed}, & \text{if } \exists t' \leq t : p(t') = c, \\ \text{unobserved}, & \text{if } \neg\text{observed}(c,t) \wedge c \in \bigcup_{c':\text{observed}} \text{adj}(c'), \\ \text{unknown}, & \text{otherwise.} \end{cases} \tag{2}$$

Task DAG. We define $\mathcal{G} = (N, E)$ as our task DAG with N, E denoting nodes and edges, respectively. We use $\text{parent}(u)$ and $\text{child}(u)$ to denote the set of parent and child nodes of u, respectively. Without loss of generality,[5] we assume a unique sink node g which denotes the goal. Each node $u \in N$ occupies a unique location in the map $l(u)$, i.e. $l : N \mapsto \mathcal{M}$ is injective.

We say node u is **visited** at timestep t if the agent is located at $l(u)$, i.e. $p(t) = l(u)$. If the agent has visited node u at some previous timestep, i.e. $\text{state}(l(u), t) = \text{observed}$, a node u has been

[5] If there are multiple goal nodes, we can append a dummy sink node.

seen.[6] Each node $u \in N$ has status$(u, t) \in \{\text{undiscovered}, \text{discovered}, \text{achieved}\}$ and type$(u) = \{\texttt{AND}, \texttt{OR}\}$ associated with it, which we define below:

$$\text{status}(u,t) = \begin{cases} \text{achieved}, & \text{if } \exists t' \le t : \text{visited}(u,t') \wedge \text{prec}(u,t'), \\ \text{discovered}, & \text{if seen}(u,t) \wedge \text{status}(u,t) \neq \text{achieved}, \\ \text{undiscovered}, & \text{if } \neg\text{seen}(u,t), \end{cases} \tag{3}$$

where prec$(u, t) =$ true if parent$(u) = \emptyset$, and when parent$(u) \neq \emptyset$,

$$\text{prec}(u,t) = \begin{cases} \forall\, u' \in \text{parent}(u) : \text{status}(u',t) = \text{achieved}, & \text{if type}(u) = \texttt{AND}, \\ \exists\, u' \in \text{parent}(u) : \text{status}(u',t) = \text{achieved}, & \text{if type}(u) = \texttt{OR}, \end{cases} \tag{4}$$

i.e. a node is only achieved when it is visited and the precondition is met. Nodes with no parents are achieved immediately upon visiting. If a node is visited before the preconditions are satisfied, the agent must satisfy the preconditions and revisit the node. The precondition requires achieving either all or at least one parent node, depending on type(u).

Upon visiting a cell $p(t) \in \mathcal{M}$, the environment provides the set of admissible movements. If a node u is discovered at $p(t)$, the environment provides information about its depth-1 neighborhood $\mathcal{N}(u) = \text{parent}(u) \cup \text{child}(u)$. The locations of nodes in $\mathcal{N}(u)$ remain unknown and must be discovered by the agent.

Throughout, we assume a movement to a neighboring cell corresponds to one timestep.[7] The task is complete when status$(g, t) =$ achieved.

Measuring Exploration and Exploitation Errors. Using the definitions from above, we can formally define pending task $\mathcal{P}(t)$:

$$\mathcal{P}(t) = \{u \in N : \text{status}(u,t) = \text{discovered} \wedge \text{prec}(u,t)\}. \tag{5}$$

Achieving tasks in $\mathcal{P}(t)$ requires **exploitation** of existing knowledge since the location is already known. However, exploitation need not be constrained to the form of traversing already seen edges, as the agent may infer a shortest path from its history, as shown in Figure 3(a).

We also formally define gain defined in Section 4:

$$\text{Gain}(t \to t+1) = 1 \Leftrightarrow \{p(t+1) \in \mathcal{T}(t)\} \vee \{\exists z \in \mathcal{T}(t) : d(p(t+1), z) < d(p(t), z)\}, \tag{6}$$

and Gain$(t \to t+1) = 0$ otherwise. Here, $d(a, b)$ is the shortest distance between the cells a and b. An action at timestep t as an error if Gain$(t \to t+1) = 0$ (Equation (6)).

We refer to the set of cells that constitute productive destinations as the **target set** $\mathcal{T}(t)$, which varies depending on the agent's current state outlined in Table 1. As depicted in Table 1, $\mathcal{T}(t)$ determines the required action from the LM agent at timestep t. Based on this, we define whether a move is a **gain** (refer to Equation (6)) if it steps into a target cell or reduces the minimum distance to at least one of the target cells. Note that the existential quantifier ensures a policy-agnostic evaluation: rather than assuming a specific strategy of the LM agent, we distinguish the map state at each timestep into four cases and determine whether the required action is exploration, exploitation or both, depicted in Table 1.

In Table 1, **Case 1** is when no tasks are pending, and the agent must explore to discover new information about the task DAG. To clarify, when the LM agent begins the game, it falls into Case 1, and both $\mathcal{P}(t)$ and $\mathcal{U}(t)$ cannot be empty at the same time. **Case 2** denotes when the goal is pending, and we assume that the only allowed strategy then is to exploit the knowledge and efficiently traverse to the goal node to finish the game. Similarly, in **Case 3** where the agent has traversed the entire map, the only viable move is to exploit its knowledge about the task DAG to complete the task.

However, when pending tasks and unseen cells both exist (**Case 4**), we assume the LM agent can either explore or exploit. In our online setting, the space of reasonable strategies is large. For instance, as depicted in Figure 3b, a path of length 7 that passes through 5 unseen cells may be more optimal for the task than a path of length 5 through seen cells. However, quantifying this tradeoff would require the full knowledge of the map and task DAG, which the agent does not have. Furthermore, since we are evaluating diverse LM agents with unknown internal reasoning, committing to any particular strategy – for example, always exploiting the nearest pending task first – and flagging deviations as errors would conflate strategic differences with genuine failures.

[6] Note that visited implies seen.

[7] This aligns with our analysis where each timestep corresponds to one API call.

Challenging Edge Cases of Exploration and Exploitation Errors Here, we list some challenging edge cases and demonstrate how our metric detects erroneous actions. All counterexamples below are instantiated on the centered 3×3 grid $\mathcal{M} = \{-1, 0, 1\}^2$ with 4-neighbor moves only. We write undirected edges as $(a, b) \leftrightarrow (c, d)$. We assume that all cells have been observed, and drop the notion of task DAGs for simplicity. All edge cases are within a no-progress segment.

Table 4: Probe a branch and back out once.

t	$p(t)$	c_t	e_t	n_t	S_t
0	$(-1, 0)$	0	0	0	0
1	$(0, 0)$	0	0	0	0
2	$(1, 0)$	0	0	0	0
3	$(0, 0)$	0	0	0	0
4	$(-1, 0)$	0	0	0	0

Explanation. This trajectory should not be penalized. It only probes a branch and backtracks once, so no cycle is formed ($c_t = 0$), no edge is traversed more than twice ($e_t = 0$), and no node is visited more than twice ($n_t = 0$). Hence the stale score never increases.

Table 5: Useful gateway revisit.

t	$p(t)$	c_t	e_t	n_t	S_t
0	$(-1, 0)$	0	0	0	0
1	$(0, 0)$	0	0	0	0
2	$(1, 0)$	0	0	0	0
3	$(0, 0)$	0	0	0	0
4	$(0, 1)$	0	0	0	0

Explanation. This trajectory should also remain error-free. The revisit to $(0, 0)$ serves as a gateway before taking a fresh branch to $(0, 1)$. Structurally, this still involves only benign backtracking: no loop is closed, and neither edge nor node reuse exceeds the allowed budget of two.

Table 6: Re-enter the same exhausted branch.

t	$p(t)$	c_t	e_t	n_t	S_t
0	$(-1, 0)$	0	0	0	0
1	$(0, 0)$	0	0	0	0
2	$(1, 0)$	0	0	0	0
3	$(0, 0)$	0	0	0	0
4	$(-1, 0)$	0	0	0	0
5	$(0, 0)$	0	1	1	2
6	$(1, 0)$	0	2	1	3

Explanation. The prefix up to $t = 4$ is still a single probe-and-return pattern and is therefore not an error. The first error appears at $t = 5$, when the trajectory reuses the edge $(-1, 0) \leftrightarrow (0, 0)$ for the third time and visits $(0, 0)$ for the third time, increasing both e_t and n_t. At $t = 6$, the edge $(0, 0) \leftrightarrow (1, 0)$ is also traversed for the third time, so e_t increases again.

Explanation. The first error occurs at $t = 4$, when the move closes a loop and increases the cyclomatic term c_t from 0 to 1. The next lap through the same cycle does not create a new independent cycle, so c_t stays constant. However, at $t = 8$ the trajectory returns to $(-1, -1)$ for the third time, increasing n_t and thus the stale score again.

Explanation. The initial oscillation up to $t = 3$ is tolerated as benign movement between two directions. The first error occurs at $t = 4$, when $(0, 0)$ is visited for the third time, increasing n_t. Subsequent steps repeatedly reuse the same corridor edges beyond twice, so e_t increases at each additional oscillation.

Explanation. The trajectory first explores one tooth of the broom and fully backtracks along the shared stem, which remains within the benign budget. The first error occurs at $t = 7$, when the

Table 7: Repeated use of the same cycle.

t	$p(t)$	c_t	e_t	n_t	S_t
0	$(-1,-1)$	0	0	0	0
1	$(0,-1)$	0	0	0	0
2	$(0,0)$	0	0	0	0
3	$(-1,0)$	0	0	0	0
4	$(-1,-1)$	1	0	0	1
5	$(0,-1)$	1	0	0	1
6	$(0,0)$	1	0	0	1
7	$(-1,0)$	1	0	0	1
8	$(-1,-1)$	1	0	1	2

Table 8: Corridor oscillation.

t	$p(t)$	c_t	e_t	n_t	S_t
0	$(0,0)$	0	0	0	0
1	$(0,1)$	0	0	0	0
2	$(0,0)$	0	0	0	0
3	$(0,-1)$	0	0	0	0
4	$(0,0)$	0	0	1	1
5	$(0,1)$	0	1	1	2
6	$(0,0)$	0	2	2	4
7	$(0,-1)$	0	3	2	5

shared stem edge $(-1,0) \leftrightarrow (0,0)$ is traversed for the third time and $(0,0)$ is visited for the third time. The final move to $(0,1)$ opens a fresh branch, so the stale score does not increase further.

C Full Results Across Prompt Modes

C.1 Models

We evaluate the following 35 models. GPT-4.1, GPT-4.1 mini, GPT-4.1 nano (OpenAI, 2025), GPT-5.4, GPT-5.4 mini, GPT-5.4 nano (OpenAI, 2026), Gemini 3.1 Pro (Google DeepMind, 2026), Gemini 3 Flash (Google DeepMind, 2025b), Gemini 3.1 Flash Lite (Google DeepMind, 2025a), Claude Opus 4.6 (Anthropic, 2026a), Claude Sonnet 4.6 (Anthropic, 2026b), Claude Haiku 4.5 (Anthropic, 2025), GPT-OSS-120B (Agarwal et al., 2025), Muse Glimmer 30B (Meta Superintelligence Lab, 2026), Gemma 4 31B (Google, 2026c), Gemma 4 26B A4B (Google, 2026b), Gemma 4 12B (Google, 2026a), Qwen 3.8 27B (Qwen Team, 2026g), Qwen 3.6 35B A3B (Qwen Team, 2026f), Qwen 3.6 27B (Qwen Team, 2026e), Qwen 3.5 397B A17B (Qwen Team, 2026b), Qwen 3.5 9B (Qwen Team, 2026d), Qwen 3.5 4B (Qwen Team, 2026c), Qwen 3.5 2B (Qwen Team, 2026a), Mistral 3 14B (Mistral AI, 2025b), Mistral 3 8B (Mistral AI, 2025d), Magistral Small (Mistral AI, 2025a), DeepSeek V4 Flash (DeepSeek-AI, 2026), Kimi K2.6 (Moonshot AI, 2026), Granite 4.1 8B (IBM, 2026), Nemotron 3 Nano 30B (NVIDIA Corporation, 2025), GLM 4.7 Flash (Z.ai, 2026), MiniMax M3 (MiniMax, 2026), Hy3 (Tencent, 2026), Hunyuan A13B (Tencent, 2025). For all models, we set the temperature as 0.

C.2 Full results in Explore-Exploit setting

Tables 10, 11, and 12 report results in our main setting, where the map is partially observable and the task DAG is unknown, so the agent must both explore and exploit. We report results aggregated over all eight prompt sets, and separately for the four prompt variants without and with reasoning. In reasoning mode, some episodes terminated before reaching the step budget because the model exhausted its output-token allowance while reasoning, without producing an action (Qwen3.6-27B: 51/108 episodes; DeepSeek-V4-Flash: 64/108 episodes). These are counted as failures. Budget used is always computed as the number of steps taken divided by the full step budget, so for these

Table 9: Comb / broom graph.

t	$p(t)$	c_t	e_t	n_t	S_t
0	$(-1, 0)$	0	0	0	0
1	$(0, 0)$	0	0	0	0
2	$(1, 0)$	0	0	0	0
3	$(1, 1)$	0	0	0	0
4	$(1, 0)$	0	0	0	0
5	$(0, 0)$	0	0	0	0
6	$(-1, 0)$	0	0	0	0
7	$(0, 0)$	0	1	1	2
8	$(0, 1)$	0	1	1	2

Table 10: Results aggregated over default and reasoning prompts (216 episodes per model). Budget used is the percentage of the step budget, averaged over all episodes (Total) or successful episodes only (Success); N/A indicates no successful episodes. Models are sorted by accuracy.

Model	Accuracy ↑	Exploration Error ↓	Exploitation Error ↓	Avg. Budget Used (Total) ↓	Avg. Budget Used (Success) ↓
Gemini 3.1 Pro (Google DeepMind, 2026)	100.00%	1.20%	0.79%	44.70%	44.70%
Claude Opus 4.6 (Anthropic, 2026a)	99.54%	1.96%	1.47%	47.48%	47.23%
Claude Sonnet 4.6 (Anthropic, 2026b)	97.22%	5.35%	2.34%	52.90%	51.55%
Claude Haiku 4.5 (Anthropic, 2025)	91.67%	8.16%	4.45%	61.27%	57.75%
GPT-5.4 (OpenAI, 2026)	89.35%	8.13%	6.00%	57.45%	52.38%
Gemini 3 Flash (Google DeepMind, 2025b)	88.43%	9.50%	4.70%	58.16%	52.68%
Kimi-K2.6 (Moonshot AI, 2026)	73.15%	11.34%	4.83%	67.90%	56.12%
GPT-4.1 (OpenAI, 2025)	66.20%	13.95%	16.85%	74.73%	61.82%
Hy3 (Tencent, 2026)	63.89%	10.14%	3.73%	73.90%	59.15%
Muse-Glimmer-30B (Meta Superintelligence Lab, 2026)	62.04%	11.11%	9.16%	77.14%	63.15%
Gemini 3.1 Flash Lite (Google DeepMind, 2025a)	50.93%	16.47%	14.94%	77.19%	55.21%
Qwen3.6-35B-A3B (Qwen Team, 2026f)	50.46%	30.52%	25.32%	80.31%	64.02%
GPT-OSS-120B (Agarwal et al., 2025)	49.07%	22.97%	23.73%	82.50%	64.34%
MiniMax-M3 (MiniMax, 2026)	47.69%	15.62%	12.47%	85.03%	69.08%
Gemma-4-31B-IT (Google, 2026c)	46.76%	17.59%	5.27%	84.33%	66.49%
Qwen3.5-397B-A17B (Qwen Team, 2026b)	43.98%	21.15%	11.12%	79.83%	60.64%
Qwen3.6-27B (Qwen Team, 2026e)	43.52%	23.27%	14.95%	59.62%	54.34%
DeepSeek-V4-Flash (DeepSeek-AI, 2026)	41.67%	11.74%	10.47%	62.87%	58.13%
Qwen3.8-27B (Qwen Team, 2026g)	41.20%	31.19%	35.35%	80.99%	53.87%
Gemma-4-26B-A4B-IT (Google, 2026b)	5.09%	54.12%	52.12%	98.02%	61.21%
Gemma-4-12B-IT (Google, 2026a)	4.63%	61.58%	63.51%	98.52%	67.96%
GPT-4.1 mini (OpenAI, 2025)	3.24%	60.92%	61.12%	98.72%	60.37%
Ministral-3-14B-Reasoning-2512 (Mistral AI, 2025c)	2.78%	58.44%	61.71%	99.23%	72.19%
Qwen3.5-9B (Qwen Team, 2026d)	2.31%	61.92%	50.88%	98.92%	70.12%
GLM-4.7-Flash (Z.ai, 2026)	1.85%	57.39%	45.43%	93.71%	62.98%
Magistral-Small-2509 (Mistral AI, 2025a)	1.39%	71.46%	57.62%	99.70%	78.09%
NVIDIA-Nemotron-3-Nano-30B-A3B-BF16 (NVIDIA Corporation, 2025)	1.39%	61.69%	48.24%	98.49%	77.81%
GPT-5.4 mini (OpenAI, 2026)	1.39%	60.53%	41.12%	99.58%	69.43%
GPT-5.4 nano (OpenAI, 2026)	1.39%	47.62%	52.72%	99.76%	82.46%
Qwen3.5-2B (Qwen Team, 2026a)	1.39%	59.49%	59.67%	96.69%	60.47%
Qwen3.5-4B (Qwen Team, 2026c)	1.39%	63.91%	36.13%	98.75%	59.06%
Hunyuan-A13B-Instruct (Tencent, 2025)	0.93%	68.51%	73.08%	98.66%	70.62%
Granite-4.1-8B (IBM, 2026)	0.46%	85.60%	91.09%	99.77%	50.88%
Ministral-3-8B-Reasoning-2512 (Mistral AI, 2025e)	0.46%	55.10%	68.83%	99.97%	93.33%
GPT-4.1 nano (OpenAI, 2025)	0.00%	83.20%	75.54%	100.00%	N/A

models the average budget used over all episodes can be lower than over successful episodes. Due to cost, we exclude the 13 API-based models from the exploit-only setting.

C.3 Full results in Exploit-only setting

To isolate exploitation, we make the entire map and the task DAG structure fully observable to the agent, so it only needs to plan and exploit its knowledge to achieve the goal. Since no exploration is required, we report only exploitation error (Tables 13, 14, and 15). As in the explore–exploit setting, some reasoning-mode episodes terminated early because the model exhausted its output-token allowance without producing an action (DeepSeek-V4-Flash: 86/108 episodes); these are counted as failures, and budget used is still normalized by the full step budget.

D Additional Experimental Details

In this section we describe the task generation process, data statistics, and prompt examples with more detail.

Table 11: Results without reasoning, aggregated over the four prompt variants (108 episodes per model). Budget used is the percentage of the step budget, averaged over all episodes (Total) or successful episodes only (Success); N/A indicates no successful episodes. Models are sorted by accuracy.

Model	Accuracy ↑	Exploration Error ↓	Exploitation Error ↓	Avg. Budget Used (Total) ↓	Avg. Budget Used (Success) ↓
Gemini 3.1 Pro (Google DeepMind, 2026)	100.00%	1.82%	1.27%	45.00%	45.00%
Claude Opus 4.6 (Anthropic, 2026a)	99.07%	2.65%	2.01%	48.96%	48.48%
Claude Sonnet 4.6 (Anthropic, 2026b)	94.44%	6.47%	2.70%	55.29%	52.66%
Claude Haiku 4.5 (Anthropic, 2025)	90.74%	9.47%	4.79%	60.13%	56.06%
Gemini 3 Flash (Google DeepMind, 2025b)	87.96%	8.78%	6.98%	59.67%	54.15%
GPT-5.4 (OpenAI, 2026)	87.04%	8.31%	7.53%	59.21%	53.13%
GPT-4.1 (OpenAI, 2025)	65.74%	12.15%	10.46%	76.86%	64.80%
Muse-Glimmer-30B (Meta Superintelligence Lab, 2026)	53.70%	11.71%	11.76%	81.13%	64.86%
Gemini 3.1 Flash Lite (Google DeepMind, 2025a)	51.85%	13.88%	11.14%	77.29%	56.21%
Kimi-K2.6 (Moonshot AI, 2026)	47.22%	17.01%	9.66%	83.37%	64.79%
DeepSeek-V4-Flash (DeepSeek-AI, 2026)	45.37%	13.97%	10.63%	84.56%	65.97%
MiniMax-M3 (MiniMax, 2026)	42.59%	14.66%	9.18%	86.21%	67.62%
Qwen3.8-27B (Qwen Team, 2026g)	41.67%	32.65%	38.18%	80.30%	52.71%
Qwen3.6-27B (Qwen Team, 2026e)	34.26%	29.42%	21.66%	86.43%	60.38%
Gemma-4-31B-IT (Google, 2026c)	32.41%	16.68%	6.90%	91.38%	73.40%
Qwen3.5-397B-A17B (Qwen Team, 2026b)	31.48%	20.02%	13.33%	89.15%	65.55%
Hy3 (Tencent, 2026)	29.63%	14.43%	7.40%	92.68%	75.30%
GPT-OSS-120B (Agarwal et al., 2025)	27.78%	26.16%	28.89%	93.05%	74.97%
Qwen3.6-35B-A3B (Qwen Team, 2026f)	17.59%	44.73%	43.80%	93.68%	64.07%
GPT-4.1 mini (OpenAI, 2025)	6.48%	52.07%	55.31%	97.43%	60.37%
Gemma-4-12B-IT (Google, 2026a)	2.78%	65.98%	71.39%	98.70%	53.27%
Gemma-4-26B-A4B-IT (Google, 2026b)	2.78%	52.03%	43.84%	98.98%	63.29%
Magistral-Small-2509 (Mistral AI, 2025a)	1.85%	69.66%	22.73%	99.72%	84.85%
GPT-5.4 mini (OpenAI, 2026)	1.85%	53.78%	24.14%	99.78%	87.88%
NVIDIA-Nemotron-3-Nano-30B-A3B-BF16 (NVIDIA Corporation, 2025)	0.93%	76.16%	76.92%	99.82%	80.70%
Qwen3.5-9B (Qwen Team, 2026d)	0.93%	80.83%	73.77%	99.38%	33.33%
Granite-4.1-8B (IBM, 2026)	0.00%	85.17%	91.75%	100.00%	N/A
Ministral-3-14B-Reasoning-2512 (Mistral AI, 2025c)	0.00%	55.82%	25.00%	100.00%	N/A
Ministral-3-8B-Reasoning-2512 (Mistral AI, 2025e)	0.00%	39.13%	19.15%	100.00%	N/A
GPT-4.1 nano (OpenAI, 2025)	0.00%	80.63%	73.22%	100.00%	N/A
GPT-5.4 nano (OpenAI, 2026)	0.00%	53.25%	57.40%	100.00%	N/A
Qwen3.5-2B (Qwen Team, 2026a)	0.00%	66.20%	60.84%	100.00%	N/A
Qwen3.5-4B (Qwen Team, 2026c)	0.00%	81.57%	0.00%	100.00%	N/A
Hunyuan-A13B-Instruct (Tencent, 2025)	0.00%	75.64%	73.63%	100.00%	N/A
GLM-4.7-Flash (Z.ai, 2026)	0.00%	69.51%	58.50%	100.00%	N/A

Table 12: Results with reasoning, aggregated over the four prompt variants (108 episodes per model). Budget used is the percentage of the step budget, averaged over all episodes (Total) or successful episodes only (Success); N/A indicates no successful episodes. Models are sorted by accuracy.

Model	Accuracy ↑	Exploration Error ↓	Exploitation Error ↓	Avg. Budget Used (Total) ↓	Avg. Budget Used (Success) ↓
Claude Opus 4.6 (Anthropic, 2026a)	100.00%	1.24%	0.90%	46.00%	46.00%
Claude Sonnet 4.6 (Anthropic, 2026b)	100.00%	4.14%	1.98%	50.51%	50.51%
Gemini 3.1 Pro (Google DeepMind, 2026)	100.00%	0.55%	0.37%	44.40%	44.40%
Kimi-K2.6 (Moonshot AI, 2026)	99.07%	1.80%	1.02%	52.43%	51.98%
Hy3 (Tencent, 2026)	98.15%	2.39%	1.86%	55.12%	54.27%
Claude Haiku 4.5 (Anthropic, 2025)	92.59%	6.87%	4.07%	62.42%	59.41%
GPT-5.4 (OpenAI, 2026)	91.67%	7.93%	4.01%	55.70%	51.67%
Gemini 3 Flash (Google DeepMind, 2025b)	88.89%	10.30%	2.16%	56.65%	51.23%
Qwen3.6-35B-A3B (Qwen Team, 2026f)	83.33%	9.90%	11.42%	66.95%	64.01%
Muse-Glimmer-30B (Meta Superintelligence Lab, 2026)	70.37%	10.49%	4.54%	73.16%	61.85%
GPT-OSS-120B (Agarwal et al., 2025)	70.37%	18.81%	19.09%	71.96%	60.15%
GPT-4.1 (OpenAI, 2025)	66.67%	15.89%	22.69%	72.59%	58.89%
Gemma-4-31B-IT (Google, 2026c)	61.11%	18.71%	4.23%	77.28%	62.83%
Qwen3.5-397B-A17B (Qwen Team, 2026b)	56.48%	22.60%	9.32%	70.51%	57.90%
MiniMax-M3 (MiniMax, 2026)	52.78%	16.62%	15.38%	83.85%	70.26%
Qwen3.6-27B (Qwen Team, 2026e)	52.78%	3.97%	0.76%	32.81%	50.42%
Gemini 3.1 Flash Lite (Google DeepMind, 2025a)	50.00%	19.01%	19.33%	77.08%	54.17%
Qwen3.8-27B (Qwen Team, 2026g)	40.74%	29.67%	31.22%	81.69%	55.05%
DeepSeek-V4-Flash (DeepSeek-AI, 2026)	37.96%	6.21%	9.85%	41.19%	48.75%
Gemma-4-26B-A4B-IT (Google, 2026b)	7.41%	56.25%	57.33%	97.07%	60.43%
Gemma-4-12B-IT (Google, 2026a)	6.48%	57.16%	53.17%	98.33%	74.26%
Ministral-3-14B-Reasoning-2512 (Mistral AI, 2025c)	5.56%	61.07%	61.90%	98.46%	72.19%
Qwen3.5-9B (Qwen Team, 2026d)	3.70%	42.87%	37.22%	98.46%	79.32%
GLM-4.7-Flash (Z.ai, 2026)	3.70%	43.17%	37.25%	87.42%	62.98%
GPT-5.4 nano (OpenAI, 2026)	2.78%	41.97%	46.22%	99.51%	82.46%
Qwen3.5-2B (Qwen Team, 2026a)	2.78%	52.26%	58.85%	93.38%	60.47%
Qwen3.5-4B (Qwen Team, 2026c)	2.78%	45.76%	38.58%	97.50%	59.06%
NVIDIA-Nemotron-3-Nano-30B-A3B-BF16 (NVIDIA Corporation, 2025)	1.85%	46.64%	39.37%	97.15%	76.36%
Hunyuan-A13B-Instruct (Tencent, 2025)	1.85%	61.07%	72.66%	97.32%	70.62%
Granite-4.1-8B (IBM, 2026)	0.93%	86.04%	90.37%	99.55%	50.88%
Magistral-Small-2509 (Mistral AI, 2025a)	0.93%	73.27%	71.96%	99.67%	64.58%
Ministral-3-8B-Reasoning-2512 (Mistral AI, 2025e)	0.93%	71.06%	78.06%	99.94%	93.33%
GPT-5.4 mini (OpenAI, 2026)	0.93%	67.32%	58.40%	99.38%	32.54%
GPT-4.1 mini (OpenAI, 2025)	0.00%	69.58%	71.73%	100.00%	N/A
GPT-4.1 nano (OpenAI, 2025)	0.00%	85.77%	87.05%	100.00%	N/A

D.1 Task Generation

We first generate each task instance programmatically with a fixed random seed, by controlling two preset controls: the size of the task DAG and the exploitation demand implied by the map. The task DAG size determines the structure of the graph by fixing the number of nodes in the DAG, for example, 4, 6, and 8 nodes for small, medium, and large size, respectively. We keep at most 3 nodes per layer (i.e. the same depth from the primitive nodes), and sample the number of prerequisites and dependencies (i.e. the edge structure) from predefined distributions. While there can be more than one primitive node of depth 0, we assume, without loss of generality, that task DAG contains exactly

Table 13: Results in the exploitation-only setting, aggregated over default and reasoning variants (216 episodes per model). Since no exploration is required, only exploitation error is reported. Budget used is the percentage of the step budget, averaged over all episodes (Total) or successful episodes only (Success); N/A indicates no successful episodes. Models are sorted by accuracy.

Model	Accuracy ↑	Exploitation Error ↓	Avg. Budget Used (Total) ↓	Avg. Budget Used (Success) ↓
Kimi-K2.6 (Moonshot AI, 2026)	87.04%	23.02%	46.06%	38.56%
MiniMax-M3 (MiniMax, 2026)	84.72%	27.56%	52.90%	44.94%
Hy3 (Tencent, 2026)	78.70%	31.52%	52.18%	39.25%
Muse-Glimmer-30B (Meta Superintelligence Lab, 2026)	74.54%	30.80%	60.47%	46.97%
Qwen3.5-397B-A17B (Qwen Team, 2026b)	66.20%	30.07%	51.91%	41.54%
Qwen3.6-35B-A3B (Qwen Team, 2026f)	57.41%	48.94%	63.53%	38.07%
Gemma-4-31B-IT (Google, 2026c)	52.78%	57.39%	70.93%	44.92%
DeepSeek-V4-Flash (DeepSeek-AI, 2026)	44.44%	28.53%	41.52%	54.93%
Qwen3.6-27B (Qwen Team, 2026e)	43.06%	52.57%	55.95%	36.81%
Qwen3.5-9B (Qwen Team, 2026d)	26.85%	64.52%	85.56%	51.71%
NVIDIA-Nemotron-3-Nano-30B-A3B-BF16 (NVIDIA Corporation, 2025)	13.89%	69.42%	90.96%	47.13%
Hunyuan-A13B-Instruct (Tencent, 2025)	12.04%	74.68%	92.69%	51.05%
Gemma-4-26B-A4B-IT (Google, 2026b)	10.19%	72.31%	95.63%	57.06%
Gemma-4-12B-IT (Google, 2026a)	9.72%	75.40%	94.65%	45.01%
Qwen3.5-4B (Qwen Team, 2026c)	7.87%	72.42%	95.81%	53.16%
GLM-4.7-Flash (Z.ai, 2026)	6.94%	71.77%	95.22%	57.09%
Magistral-Small-2509 (Mistral AI, 2025a)	6.48%	82.45%	96.79%	50.47%
Ministral-3-14B-Reasoning-2512 (Mistral AI, 2025c)	4.63%	72.65%	97.90%	54.67%
Ministral-3-8B-Reasoning-2512 (Mistral AI, 2025e)	3.70%	72.59%	98.56%	61.07%
Qwen3.8-27B (Qwen Team, 2026g)	3.70%	90.42%	97.46%	31.32%
Qwen3.5-2B (Qwen Team, 2026a)	1.39%	74.23%	85.87%	75.84%
Granite-4.1-8B (IBM, 2026)	0.00%	89.83%	100.00%	N/A

Table 14: Results in the exploitation-only setting without reasoning (108 episodes per model). Budget used is the percentage of the step budget, averaged over all episodes (Total) or successful episodes only (Success); N/A indicates no successful episodes. Models are sorted by accuracy.

Model	Accuracy ↑	Exploitation Error ↓	Avg. Budget Used (Total) ↓	Avg. Budget Used (Success) ↓
Kimi-K2.6 (Moonshot AI, 2026)	75.00%	30.89%	63.01%	50.68%
MiniMax-M3 (MiniMax, 2026)	70.37%	35.39%	70.24%	57.70%
DeepSeek-V4-Flash (DeepSeek-AI, 2026)	68.52%	28.98%	72.22%	59.46%
Hy3 (Tencent, 2026)	57.41%	41.98%	74.73%	55.97%
Muse-Glimmer-30B (Meta Superintelligence Lab, 2026)	56.48%	35.95%	79.53%	63.76%
Qwen3.5-397B-A17B (Qwen Team, 2026b)	48.15%	35.80%	61.96%	48.33%
Qwen3.6-27B (Qwen Team, 2026e)	24.07%	62.47%	88.29%	51.35%
Qwen3.6-35B-A3B (Qwen Team, 2026f)	16.67%	61.88%	93.26%	59.56%
Gemma-4-31B-IT (Google, 2026c)	15.74%	71.27%	93.04%	55.80%
Gemma-4-12B-IT (Google, 2026a)	8.33%	73.83%	96.08%	53.00%
Gemma-4-26B-A4B-IT (Google, 2026b)	8.33%	67.67%	97.23%	66.76%
Magistral-Small-2509 (Mistral AI, 2025a)	5.56%	78.86%	97.76%	59.68%
Qwen3.8-27B (Qwen Team, 2026g)	5.56%	87.78%	96.20%	31.62%
Ministral-3-14B-Reasoning-2512 (Mistral AI, 2025c)	4.63%	62.18%	98.70%	72.00%
Qwen3.5-9B (Qwen Team, 2026d)	4.63%	76.24%	97.77%	51.85%
Ministral-3-8B-Reasoning-2512 (Mistral AI, 2025e)	1.85%	56.87%	99.67%	82.21%
Qwen3.5-2B (Qwen Team, 2026a)	0.93%	81.11%	99.58%	55.17%
Qwen3.5-4B (Qwen Team, 2026c)	0.93%	82.14%	99.76%	73.81%
Hunyuan-A13B-Instruct (Tencent, 2025)	0.93%	84.63%	99.35%	29.82%
Granite-4.1-8B (IBM, 2026)	0.00%	88.91%	100.00%	N/A
NVIDIA-Nemotron-3-Nano-30B-A3B-BF16 (NVIDIA Corporation, 2025)	0.00%	81.52%	100.00%	N/A
GLM-4.7-Flash (Z.ai, 2026)	0.00%	78.41%	100.00%	N/A

one goal node. All prerequisite edges point from shallower nodes to deeper nodes to ensure that the sampled graph is acyclic.

To prevent agents from exploiting semantics in node names, each node is named with a random four-character visible token, e.g. D7UX and 9J7T, rather than a meaningful name. Note that using simple alphanumerics such as A, B, C, D or 1, 2, 3, 4 can also erroneously inject information regarding the precedence. If the sampled graph contains any node disconnected from the goal node, the goal's prerequisite is repaired so that every node remains relevant to task completion.

After the task DAG is sampled, we map it onto a 2D grid. The 2D grid's size is determined with the ratio of the number of task nodes in the task DAG to the number of cells in the grid – i.e. setting a lower density produces a larger map, unless the map size is fixed a priori.

The exploitation demand controls how the 2D grid map is generated, and we use two knobs to control the exploitation demand: (i) map density and (ii) corridor widths. Lower map density implies that the task nodes are placed in a more sparse fashion, implicitly increasing the distance between the task nodes, and hence requires more exploration. Similarly, increasing the corridor width not only increases the effective map size but also generates more admissible moves on each cell, compared to having long narrow paths. When generating the maps, we use densities of 0.1, 0.25, and 0.4 and corridor widths of 2–3, 1–3, and 1 cells for low, medium, and high exploitation demand, respectively. For corridor widths, we use 1, 1-3, 2-3 cells for low, medium, and high exploration. The cell in which

Table 15: Results in the exploitation-only setting with reasoning (108 episodes per model). Budget used is the percentage of the step budget, averaged over all episodes (Total) or successful episodes only (Success); N/A indicates no successful episodes. Models are sorted by accuracy.

Model	Accuracy ↑	Exploitation Error ↓	Avg. Budget Used (Total) ↓	Avg. Budget Used (Success) ↓
Hy3 (Tencent, 2026)	100.00%	2.91%	29.64%	29.64%
MiniMax-M3 (MiniMax, 2026)	99.07%	10.51%	35.56%	35.88%
Kimi-K2.6 (Moonshot AI, 2026)	99.07%	3.72%	29.12%	29.38%
Qwen3.6-35B-A3B (Qwen Team, 2026f)	98.15%	7.24%	33.80%	34.42%
Muse-Glimmer-30B (Meta Superintelligence Lab, 2026)	92.59%	19.89%	41.41%	36.73%
Gemma-4-31B-IT (Google, 2026c)	89.81%	29.34%	48.82%	43.02%
Qwen3.5-397B-A17B (Qwen Team, 2026b)	84.26%	20.11%	41.87%	37.66%
Qwen3.6-27B (Qwen Team, 2026e)	62.04%	5.85%	23.61%	31.17%
Qwen3.5-9B (Qwen Team, 2026d)	49.07%	47.50%	73.35%	51.69%
NVIDIA-Nemotron-3-Nano-30B-A3B-BF16 (NVIDIA Corporation, 2025)	27.78%	54.56%	81.92%	47.13%
Hunyuan-A13B-Instruct (Tencent, 2025)	23.15%	62.70%	86.04%	51.90%
DeepSeek-V4-Flash (DeepSeek-AI, 2026)	20.37%	24.01%	10.82%	39.71%
Qwen3.5-4B (Qwen Team, 2026c)	14.81%	61.59%	91.85%	51.87%
GLM-4.7-Flash (Z.ai, 2026)	13.89%	64.27%	90.43%	57.09%
Gemma-4-26B-A4B-IT (Google, 2026b)	12.04%	77.15%	94.02%	50.35%
Gemma-4-12B-IT (Google, 2026a)	11.11%	77.06%	93.22%	39.02%
Magistral-Small-2509 (Mistral AI, 2025a)	7.41%	86.07%	95.82%	43.57%
Ministral-3-8B-Reasoning-2512 (Mistral AI, 2025e)	5.56%	88.53%	97.45%	54.02%
Ministral-3-14B-Reasoning-2512 (Mistral AI, 2025c)	4.63%	83.53%	97.10%	37.35%
Qwen3.5-2B (Qwen Team, 2026a)	1.85%	63.55%	72.15%	86.17%
Qwen3.8-27B (Qwen Team, 2026g)	1.85%	93.00%	98.71%	30.43%
Granite-4.1-8B (IBM, 2026)	0.00%	90.76%	100.00%	N/A

the LM agent starts do not contain any task node, and corridors are generated between the starting position and task nodes to enforce that the traversable region is fully connected.

Beyond the task DAG and exploitation presets, the environment generator controls the spatial layout and budget per episode. First, the aspect ratio of 2D grid map specifies the target width-to-height ratio of the grid map, after the 2D grid map size is determined from the node budget and exploitation demand. The generator selects integer dimensions that best match this ratio. A budget parameter α defines the hidden interaction budget as the following:

$$B = \alpha|\mathcal{O}|, \tag{7}$$

where $|\mathcal{O}|$ denotes the number of traversable cells in the generated map. Importantly, α does not alter the sampled DAG or the grid geometry itself; it simply sets the maximum number of turns the LM agent can take to solve the task. In our setup, we fix the aspect ratio of 2D grid map as 1 and a budget parameter as 3.

Finally, we use three categorical distributions to determine the logical complexity of the symbolic task graph. For each non-primitive node, we use `option_count_probs` to control the number of alternative prerequisite sets and therefore the amount of `OR`-type branching, while `dependency_count_probs` controls the number of parents within each prerequisite set and therefore the `AND`-type arity of each option. The parameter `goal_dependency_count_probs` plays the same role for the unique goal node, determining how many predecessors must be jointly satisfied at the top of the graph. Parent nodes are sampled by controlling parent-depth bias: a candidate parent at depth d for a child at depth D receives weight proportional to $\exp(-\beta((D-1)-d))$, where β is the bias parameter, where larger β favors dependencies from more recent layers. We use $\beta = 1$ (i.e., `parent_depth_bias`$= 1$) for all tasks. For easy, medium, and hard maps, respectively, we set `option_count_probs` to $\{1:1.0\}$, $\{1:0.8, 2:0.2\}$, and $\{1:0.6, 2:0.4\}$, and set both `dependency_count_probs` and `goal_dependency_count_probs` to uniform distributions over $\{1,2\}$, $\{1,2\}$, and $\{1,2,3\}$.

D.2 Statistics on Experiments

We report the number of episodes and seed configurations used in each experiment.

Main Experiment. For the main experiment, we use procedurally generated maps parameterized by three exploitation demand (`low`, `medium`, `high`) and three task-DAG size (`small`, `medium`, `large`), yielding $3 \times 3 = 9$ distinct map configurations. Each configuration is run with 3 seeds (seeds 0, 1, 2), producing 27 episodes per prompt set. We evaluate 8 prompt sets (4 base types $\times$ 2 reasoning modes: `default`, `default-balance`, `default-exploration`, `default-exploitation`, `reasoning`, `reasoning-balance`, `reasoning-exploration`, `reasoning-exploitation`), resulting in $8 \times 27 = 216$ total episodes per model. The harness engineering experiments use the

same procedurally generated dataset of 9 map configurations with 3 seeds per configuration, identical to the main experiment.

Semantic Experiment. For the semantic experiments, we use 4 hand-crafted custom maps (custom-201 through custom-204), each available in both a semantic variant (with meaningful state names) and a symbolic variant (with abstract identifiers). Each map is evaluated with 5 seeds (seeds 0–4), yielding $4 \times 5 = 20$ episodes per prompt-variant combination. We test 2 prompt sets (`default`, `reasoning`) across both semantic and symbolic variants, for a total of $2 \times 2 \times 20 = 80$ episodes per model.

D.3 Prompts

Each system prompt consists of three parts: (1) an environment description, (2) an optional strategy prompt, and (3) an action format specification. Parts (1) and (3) are identical across all four variants. The four variants differ only in the strategy prompt (part 2). Below we show the full system prompt for each variant, with the strategy sentence highlighted in yellow. We also provide example turns illustrating the observation format.

- **Base Prompt.** No strategy sentence is included. The agent receives only the environment description and action format, leaving the exploration–exploitation tradeoff entirely to the model's own reasoning.

- **Exploration Prompt.** A single strategy sentence is inserted between the environment description and the action format, instructing the agent to prioritize visiting unvisited cells.

- **Exploitation Prompt.** The strategy sentence instructs the agent to prioritize achieving already-discovered states whose prerequisites are satisfied.

- **Balance Prompt.** The strategy sentence combines both the exploration and exploitation definitions and asks the agent to choose the balance that minimizes total steps.

Example Turns. At each turn, the agent receives an observation and must produce a single JSON action. Below are two example turns: one where the agent finds nothing, and one where a state is discovered.

Example Observation (Empty Cell)

```
OBSERVATION: You are at [7, 0].  You found nothing here.  Available
directions:  up
```

Example Agent Response

```
{"action":"up"}
```

Example Observation (State Discovery)

```
OBSERVATION: You are at [0, 1].  You discovered state U_02.  U_02
has no prerequisites and is immediately activated!  U_02 has
ancestors:  R_01, G_00.  Available directions:  up, down, right
```

System Prompt (Base)

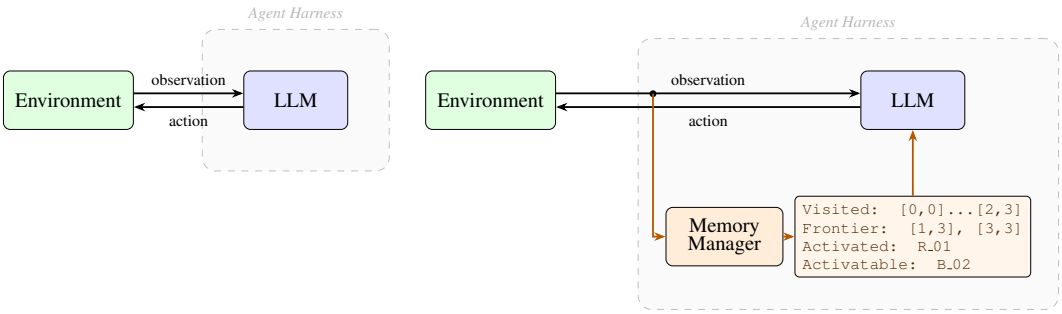
```
You are controlling an agent in a partially observed symbolic grid
environment.  Your objective is to activate the goal state.  At
each step, you are given your current position, the directions you
can legally move, and any newly discovered symbolic states at your
current cell.  Newly discovered states may include prerequisite
information and ancestor hints.  A state can be activated when you
are on its cell and its prerequisites are satisfied.  The full map,
hidden budget, and undiscovered states are not available to you.
Reply with exactly one JSON object containing one valid action from
available_directions like this:  {"action":"up"}, {"action":"down"},
{"action":"left"}, {"action":"right"}
```

System Prompt (Exploration)

```
You are controlling an agent in a partially observed symbolic grid
environment.  Your objective is to activate the goal state.  At
each step, you are given your current position, the directions you
can legally move, and any newly discovered symbolic states at your
current cell.  Newly discovered states may include prerequisite
information and ancestor hints.  A state can be activated when you
are on its cell and its prerequisites are satisfied.  The full
map, hidden budget, and undiscovered states are not available
to you.  Prioritize exploration when deciding where to move.
Treat exploration as deliberately moving toward cells you have
not visited yet and roaming to uncover cells and symbolic states
that you have not discovered yet. Reply with exactly one JSON
object containing one valid action from available_directions like
this:  {"action":"up"}, {"action":"down"}, {"action":"left"},
{"action":"right"}
```

System Prompt (Exploitation)

```
You are controlling an agent in a partially observed symbolic grid
environment.  Your objective is to activate the goal state.  At
each step, you are given your current position, the directions you
can legally move, and any newly discovered symbolic states at your
current cell.  Newly discovered states may include prerequisite
information and ancestor hints.  A state can be activated when you
are on its cell and its prerequisites are satisfied.  The full map,
hidden budget, and undiscovered states are not available to you.
Prioritize exploitation when deciding where to move. Among the
symbolic states you have already discovered, first target states
whose prerequisites are already satisfied, and move along the
shortest available path to activate them. Reply with exactly one
JSON object containing one valid action from available_directions
like this:  {"action":"up"}, {"action":"down"}, {"action":"left"},
{"action":"right"}
```



(a) Baseline (b) Harness Engineering

Figure 7: Harness Design Comparison. (a) In the baseline, the agent harness passes observations directly to the LLM and returns actions as in Yao et al. (2023). (b) With harness engineering, each observation is also routed through a rule-based memory manager that produces a structured summary (visited cells, frontier, activated and activatable states, ...) which is injected into the LLM prompt. The information content is identical but the relevant information is highlighted explicitly through harness engineering.

System Prompt (Balance)

```
You are controlling an agent in a partially observed symbolic grid
environment. Your objective is to activate the goal state. At
each step, you are given your current position, the directions you
can legally move, and any newly discovered symbolic states at your
current cell. Newly discovered states may include prerequisite
information and ancestor hints. A state can be activated when you
are on its cell and its prerequisites are satisfied. The full map,
hidden budget, and undiscovered states are not available to you.
Balance exploration and exploitation when deciding where to move.
Treat exploration as deliberately moving toward cells you have
not visited yet and roaming to uncover cells and symbolic states
that you have not discovered yet. Treat exploitation as targeting
already discovered symbolic states whose prerequisites are already
satisfied and moving along the shortest available path to activate
them. Choose the balance between these two behaviors based on which
actions are most likely to solve the task in the fewest steps.
Reply with exactly one JSON object containing one valid action from
available_directions like this: {"action":"up"}, {"action":"down"},
{"action":"left"}, {"action":"right"}
```

E Different Exploration and Exploitation Behaviors beyond success

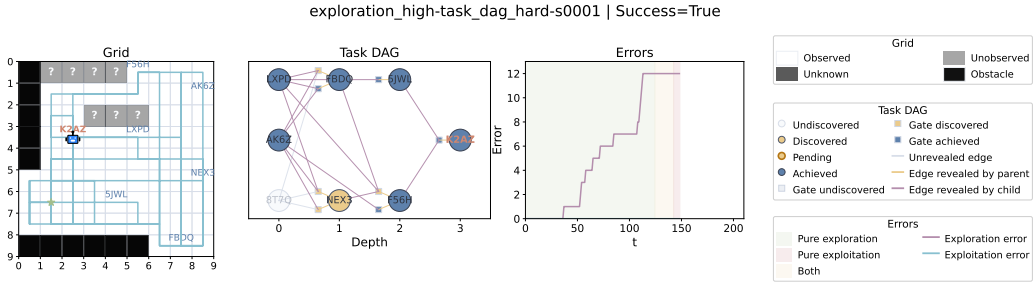
To illustrate that task success alone cannot distinguish how an LM agent solves a task, we compare Hy3 and GPT-OSS-120B on the same map with the same default prompt. Both models achieve the goal within the budget of 210 steps, yet exhibit markedly different behaviors (Table 16 and Figure 8).

Hy3 explores somewhat noisily, incurring an exploration error of 8.1%, but exploits cleanly without any exploitation error, and reaches the goal in 148 steps. In contrast, GPT-OSS-120B explores without any exploration error, but exploits poorly. It discovers the goal node at step 22 and satisfies all of its prerequisites by step 78, after which the goal is pending and the only productive action is to return to it (Case 2 in Table 1). Nevertheless, the agent spends another 110 steps wandering before reaching the goal at step 188, incurring an exploitation error of 47.3%.

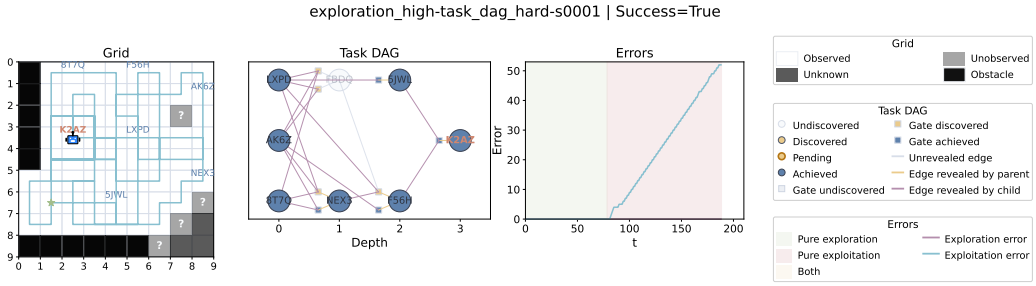
This example shows that two trajectories with the same outcome can be inefficient in opposite ways: Hy3 is limited by exploration, whereas GPT-OSS-120B is limited by exploitation. Our metric separates the two failure modes, which success rate alone cannot capture.

Table 16: Hy3 and GPT-OSS-120B on the same map and prompt. Both succeed, but Hy3 incurs only exploration errors, whereas GPT-OSS-120B incurs only exploitation errors.

Model	Steps / Budget	Exploration Error	Exploitation Error
Hy3	148 / 210	8.1% (12/148)	0.0% (0/24)
GPT-OSS-120B	188 / 210	0.0% (0/188)	47.3% (52/110)



(a) Hy3



(b) GPT-OSS-120B

Figure 8: Trajectories of Hy3 and GPT-OSS-120B on the same map with the exploration prompt without reasoning. Each panel shows the final map state, the task DAG, and the errors over timesteps.

F Harness Engineering

Figure 7 illustrates the information flow in the baseline agent harness and explicit harness engineering. Specifically, we inject a **structured memory summary** alongside the observation at every turn, which is a rule-based memory manager that aggregates past observations into a fixed-format summary and appends it to the turn prompt. Crucially, the memory summary contains *no new information*: every field is derivable from the history of observations the agent has already received. The engineering lies entirely in *how* the information is organized and presented.

The memory summary consists of the following components:

- **Learned coordinate system:** inferred from movement outcomes (e.g., "up = y+1, right = x+1").
- **Known goal state:** the goal state name, once discovered.
- **Visited cells:** cumulative list of all cells the agent has stepped on.
- **Traversable frontier:** cells known to be reachable (observed as neighbors) but not yet visited, directly supporting exploration.
- **Obstacle cells:** cells inferred to be impassable.
- **Discovered states:** each discovered state with its position, prerequisite conditions, and successor states.

- **Activated / activatable states:** which states have been activated, and which states currently have all prerequisites satisfied and are ready for activation, directly supporting exploitation.

The system prompt is updated to inform the model that this summary is available (lines highlighted in yellow):

System Prompt (with Harness Engineering)

```
You are controlling an agent in a partially observed symbolic grid
environment.  Your objective is to activate the goal state.  At
each step, you are given your current observation that contains
your current position, the directions you can legally move,
and any newly discovered symbolic states at your current cell.
Also, you are given a summary of your explored map, visited
cells, reachable frontier cells, discovered states, activated
states, and prerequisite relations. Newly discovered states may
include prerequisite information and ancestor hints.  A state
can be activated when you are on its cell and its prerequisites
are satisfied.  The full map, hidden budget, and undiscovered
states are not available to you.  Reply with exactly one JSON
object containing one valid action from available_directions like
this:  {"action":"up"}, {"action":"down"}, {"action":"left"},
{"action":"right"}
```

Below is an example turn prompt after 12 steps. The observation (top) is identical to the baseline; the memory summary (bottom) is the injected harness.

Example Observation with Harness Engineering (Step 12)

```
OBSERVATION: You are at [2, 3].  You found nothing here.  Your
available action is up, down, left, or right.  You spent 12 steps.

From your movements so far, coordinate system:  [x, y] where up=y+1,
down=y-1, left=x-1, right=x+1.  The goal state is G_00.  You have
visited [0, 0], [0, 1], [1, 1], [1, 2], [2, 2], [2, 3].  You have
not visited these cells yet, but you know you can pass through them:
[1, 3], [3, 3].  You know you cannot pass through these cells:  [2,
4].  You discovered R_01 at [0, 0].  It is already activated.  Once
it is activated, you can next pursue B_02.  You discovered B_02
at [1, 2].  It is not activated yet.  To activate it, you should
find R_01 first.  Once it is activated, you can next pursue G_00.
Activated states:  R_01.  The discovered states whose prerequisites
are already satisfied:  B_02.
```

In this example, the baseline agent would need to recall from conversation history that B_02's prerequisite R_01 has already been activated and that B_02 is therefore ready to achieve. With the harness, this information is stated explicitly ("The discovered states whose prerequisites are already satisfied: B_02"), allowing the model to exploit this information without relying on its memory retrieval. Similarly, the frontier cells ([1, 3], [3, 3]) are explicitly listed, guiding exploration toward unvisited but reachable cells.

G Semantic Experiment

For the semantic information injection experiments, we generate four task instances based on pasta-cooking scenarios. As shown in Figure 9, the task DAG generation is motivated from real-world cooking tasks. To prepare the final dish, the agent must visit all required cells on the 2D grid to find the ingredients and perform intermediate steps.

Observation Format. The grid map, task DAG structure, and observation format remain identical between the symbolic and semantic experiments. The only difference lies in the *task DAG's node names* presented to the agent in each observation. In symbolic experiments, state names are replaced with randomly generated four-character alphanumeric codes (e.g., `B4KD`, `H2NZ`, `Q7M2`), designed

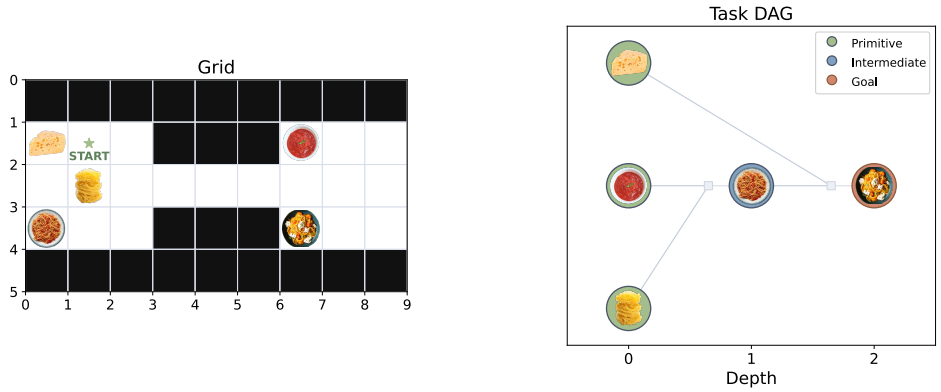


Figure 9: An oracle view of the 2D grid map and the corresponding task DAG for experiments with semantic information injection. This example illustrates cooking tomato pasta with cheese. First, tomato pasta is made from pasta and tomato sauce. Second, tomato pasta with cheese is made by adding cheese to the tomato pasta. In this case, the LM agent may overly rely on semantic priors. For example, it might erroneously assume that the pasta sauce and cheese must be placed in proximity.

to carry no semantic meaning. In the semantic experiments, however, the task nodes are labeled with meaningful cooking-related names (e.g., `Pasta`, `Tomato Pasta`, `Tomato Pasta with Cheese`).

Below we show two example observations from the same 2D grid cell (i.e. what the environment returns to the LM agent), illustrating how the agent perceives a discovered state under each state.

Observation — Symbolic Condition

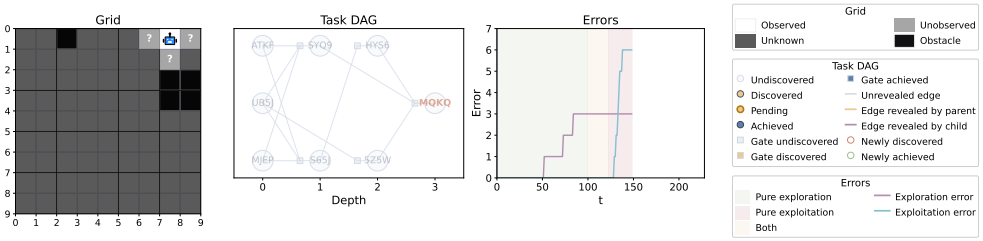
```
OBSERVATION: You are at [1, 2].  You found B4KD which is now
activated.  Now, you can go to H2NZ. Your available action is down,
left, right, or up.  You spent 21 steps.
```

Observation — Semantic Condition

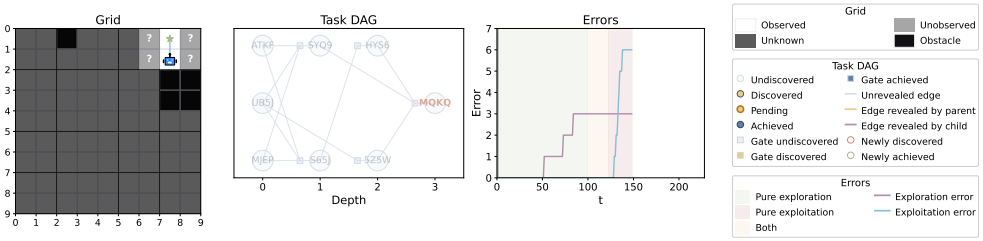
```
OBSERVATION: You are at [1, 2].  You found Pasta which is now
activated.  Now, you can go to Tomato Pasta.  Your available action
is down, left, right, or up.  You spent 1 steps.
```

H Examples of LM Agent's Runs in Our Environments

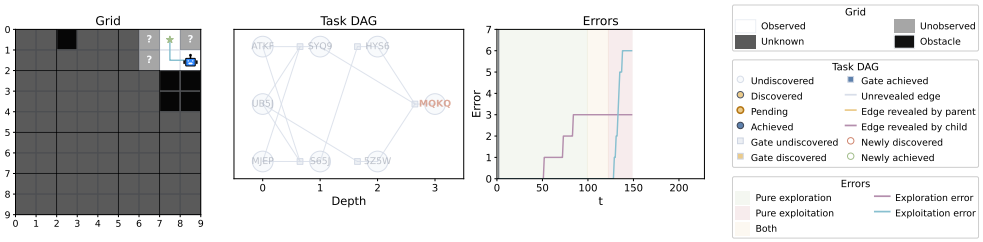
We visualize the action trajectories of Claude Haiku 4.5, Claude Sonnet 4.6, Claude Opus 4.6, Gemini 3 Flash, Gemini 3.1 Flash Lite, Gemini 3.1 Pro, GPT-4.1 mini, GPT-4.1, GPT-5.4 mini, GPT-5.4, and GPT-OSS-120B, and our metric values in Figures 10 to 42 over each timestep.



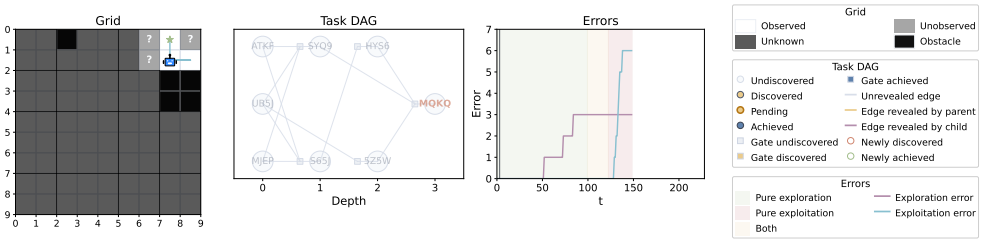
(a) Claude Haiku 4.5, $t = 0$



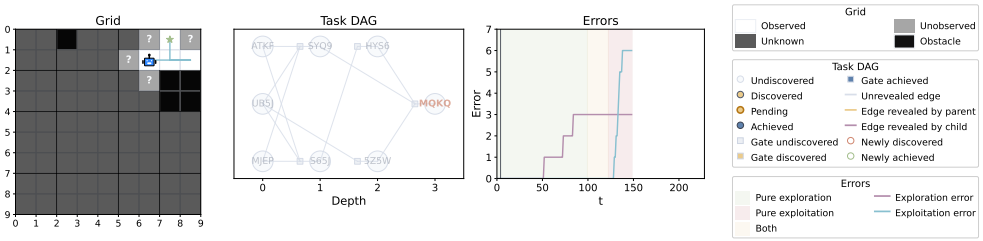
(b) Claude Haiku 4.5, $t = 1$



(c) Claude Haiku 4.5, $t = 2$

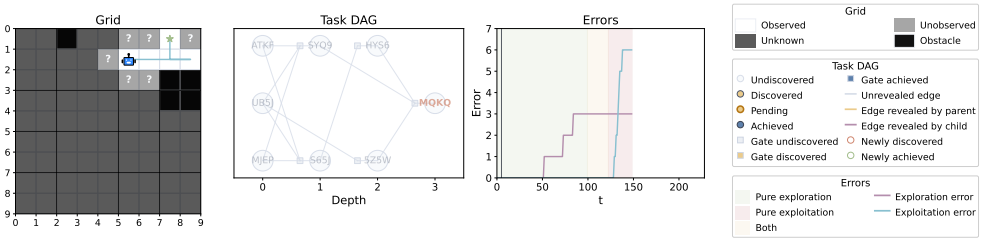


(d) Claude Haiku 4.5, $t = 3$

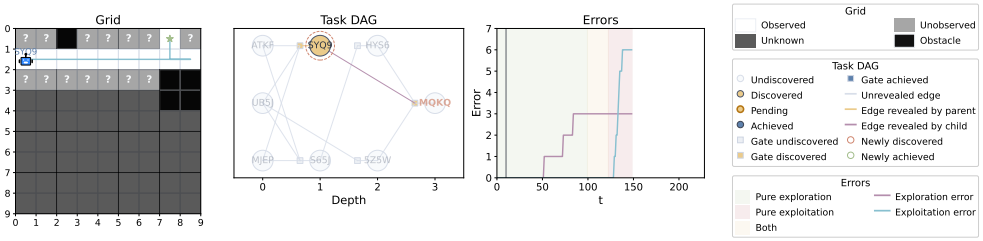


(e) Claude Haiku 4.5, $t = 4$

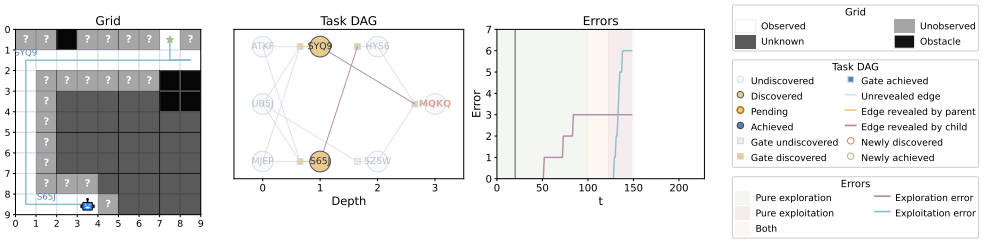
Figure 10: Results of action trajectories and metric values over each timestep for Claude Haiku 4.5.



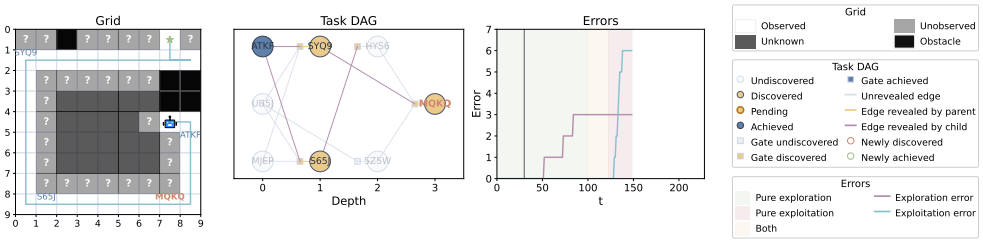
(a) Claude Haiku 4.5, $t = 5$



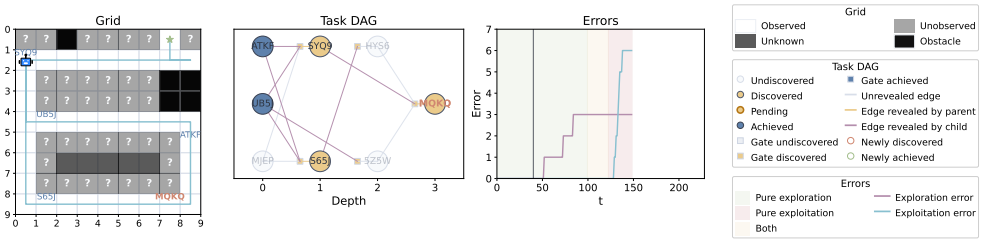
(b) Claude Haiku 4.5, $t = 10$



(c) Claude Haiku 4.5, $t = 20$

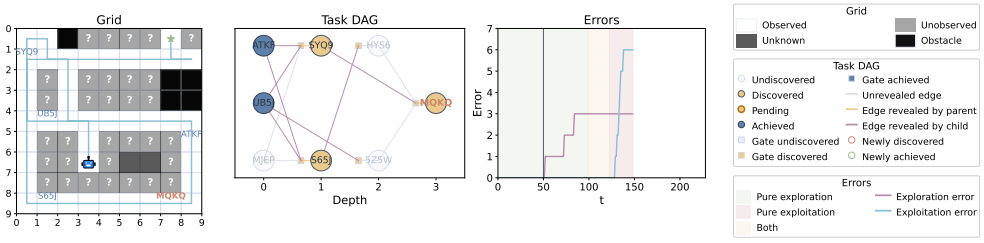


(d) Claude Haiku 4.5, $t = 30$

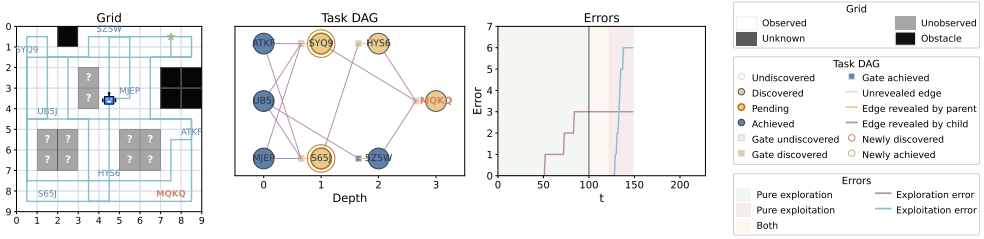


(e) Claude Haiku 4.5, $t = 40$

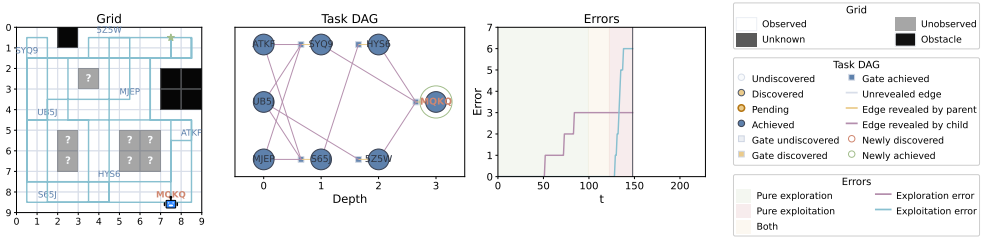
Figure 11: Results of action trajectories and metric values over each timestep for Claude Haiku 4.5.



(a) Claude Haiku 4.5, $t = 50$

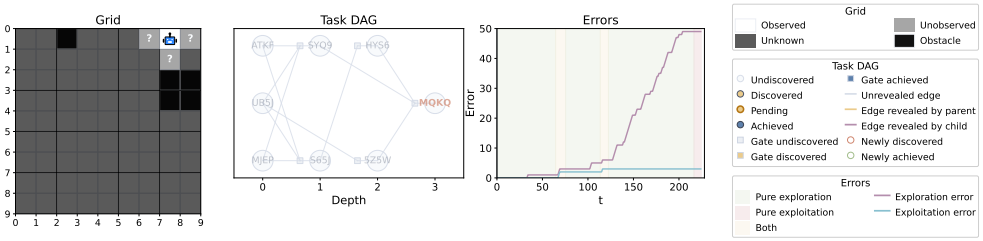


(b) Claude Haiku 4.5, $t = 100$

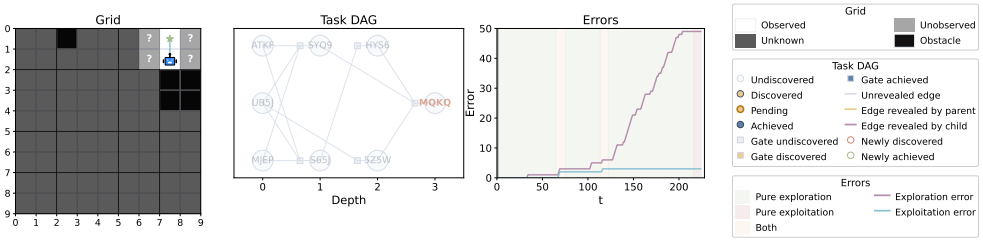


(c) Claude Haiku 4.5, $t = 148$

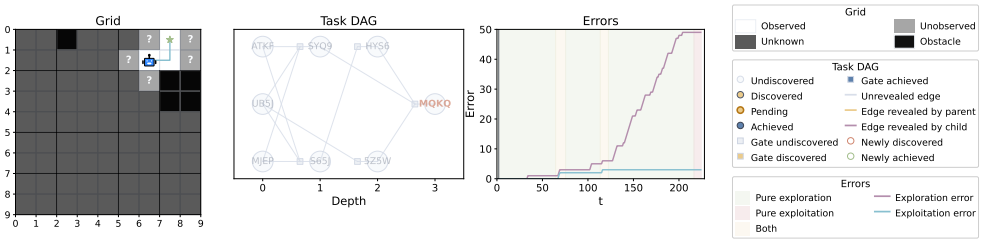
Figure 12: Results of action trajectories and metric values over each timestep for Claude Haiku 4.5.



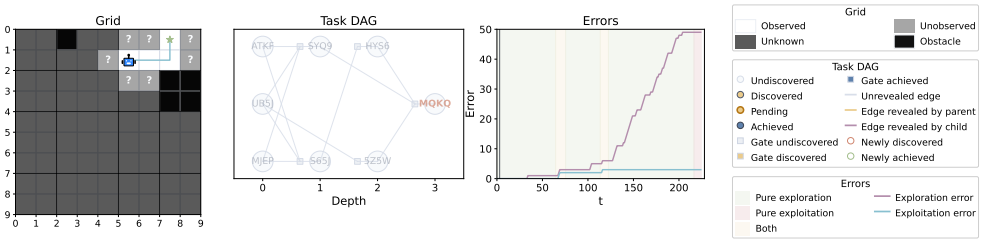
(a) Claude Sonnet 4.6, $t = 0$



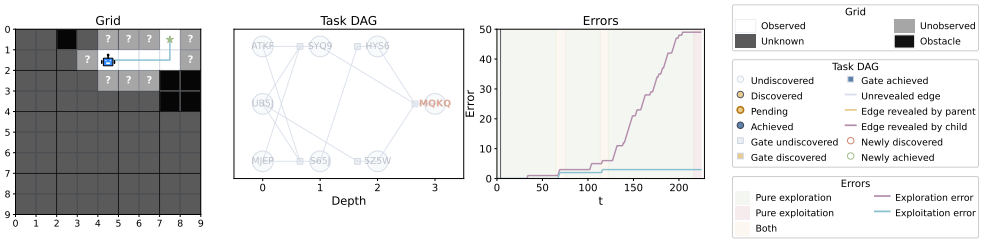
(b) Claude Sonnet 4.6, $t = 1$



(c) Claude Sonnet 4.6, $t = 2$

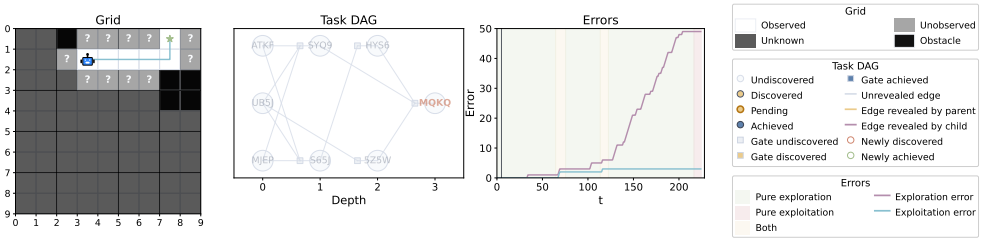


(d) Claude Sonnet 4.6, $t = 3$

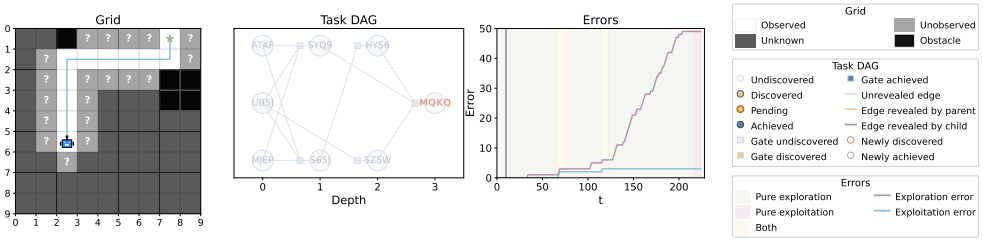


(e) Claude Sonnet 4.6, $t = 4$

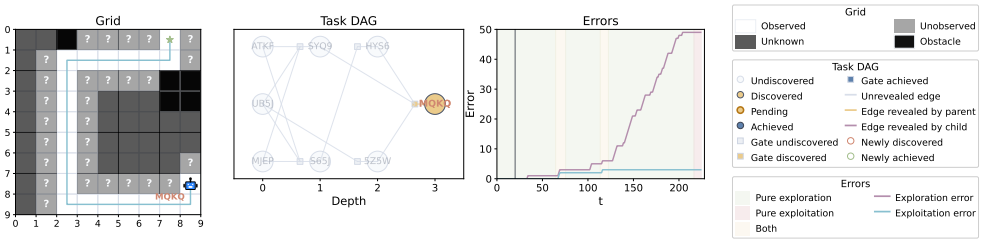
Figure 13: Results of action trajectories and metric values over each timestep for Claude Sonnet 4.6.



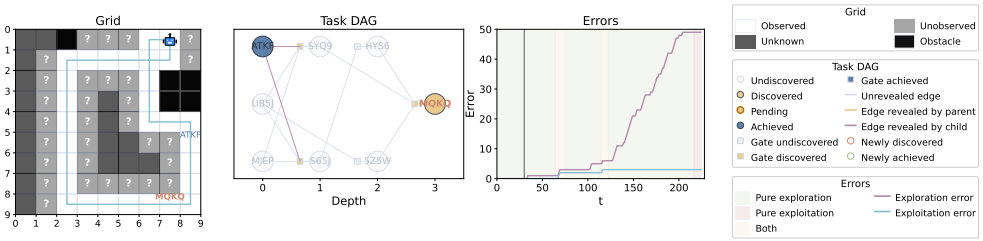
(a) Claude Sonnet 4.6, $t = 5$



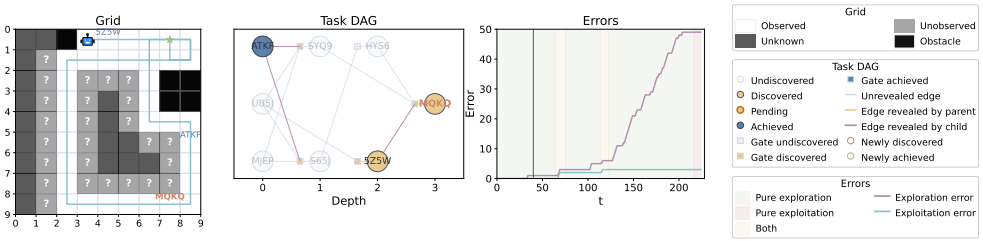
(b) Claude Sonnet 4.6, $t = 10$



(c) Claude Sonnet 4.6, $t = 20$

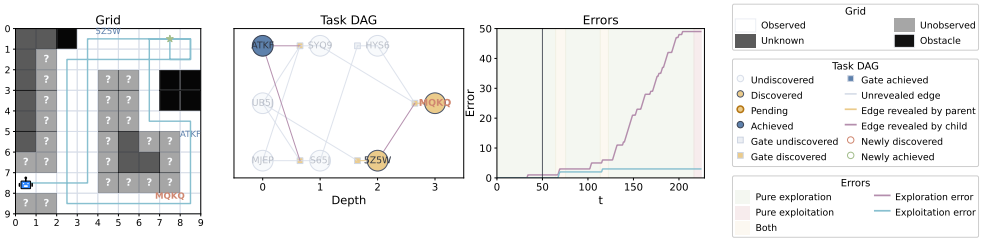


(d) Claude Sonnet 4.6, $t = 30$

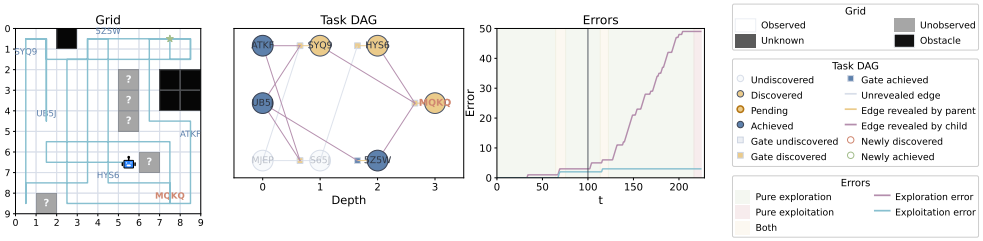


(e) Claude Sonnet 4.6, $t = 40$

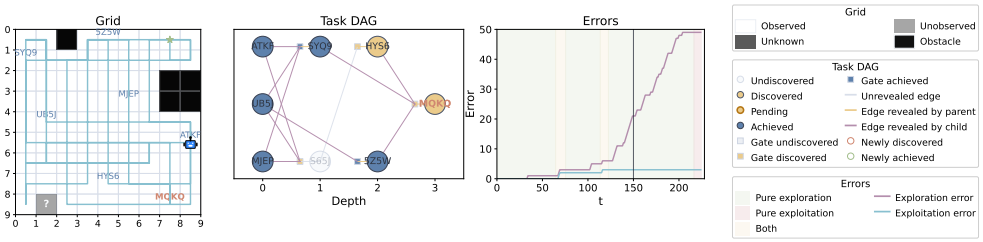
Figure 14: Results of action trajectories and metric values over each timestep for Claude Sonnet 4.6.



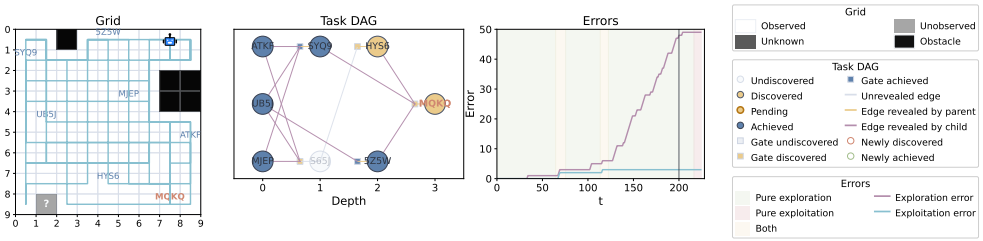
(a) Claude Sonnet 4.6, $t = 50$



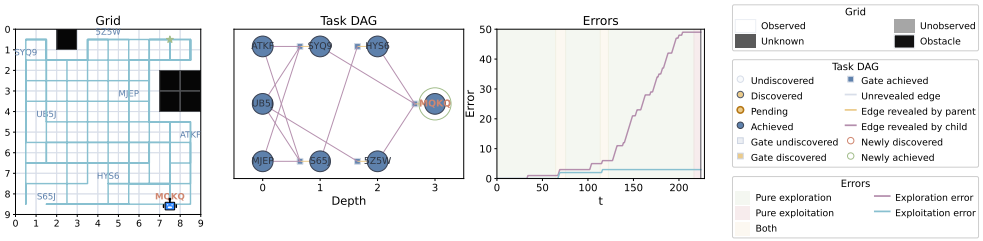
(b) Claude Sonnet 4.6, $t = 100$



(c) Claude Sonnet 4.6, $t = 150$

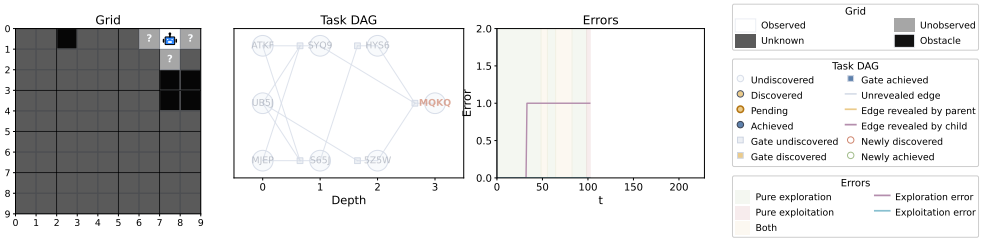


(d) Claude Sonnet 4.6, $t = 200$

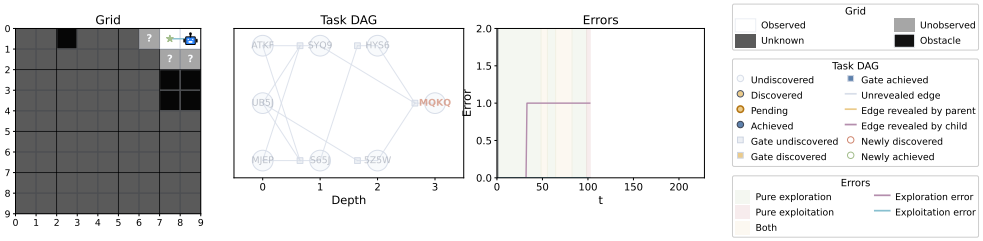


(e) Claude Sonnet 4.6, $t = 224$

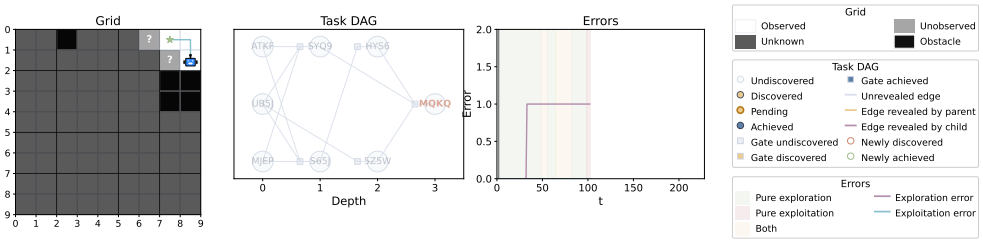
Figure 15: Results of action trajectories and metric values over each timestep for Claude Sonnet 4.6.



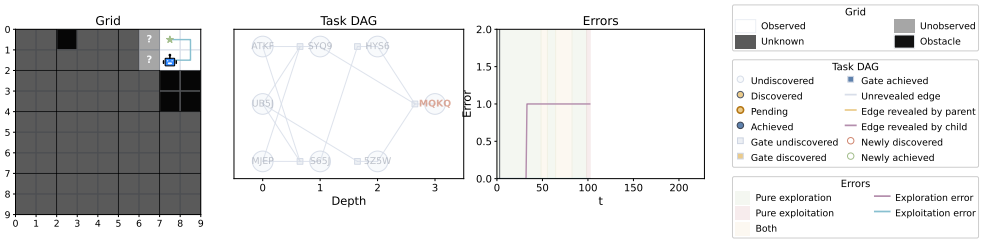
(a) Claude Opus 4.6, $t = 0$



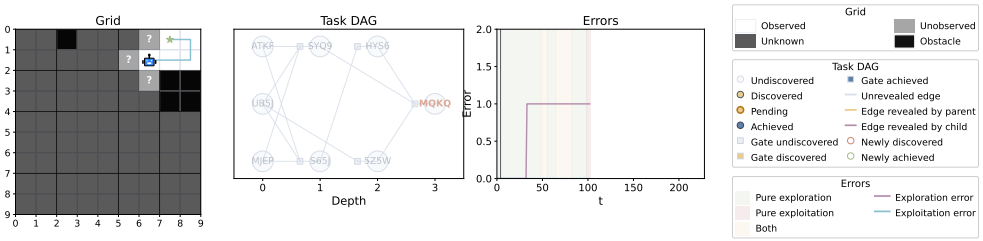
(b) Claude Opus 4.6, $t = 1$



(c) Claude Opus 4.6, $t = 2$

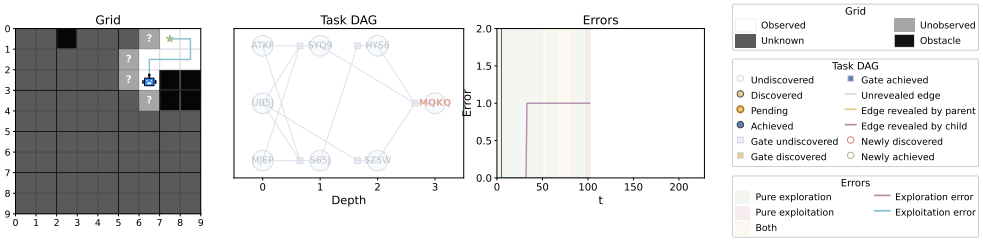


(d) Claude Opus 4.6, $t = 3$

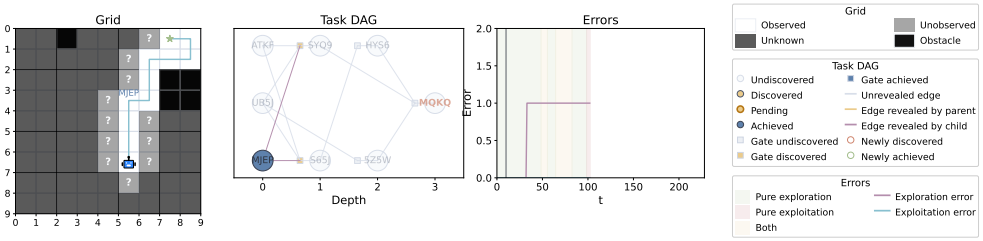


(e) Claude Opus 4.6, $t = 4$

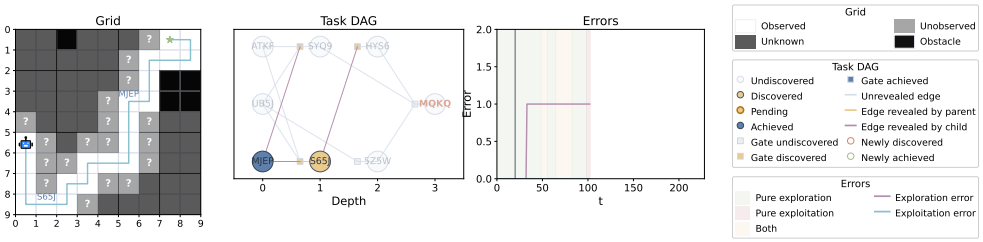
Figure 16: Results of action trajectories and metric values over each timestep for Claude Opus 4.6.



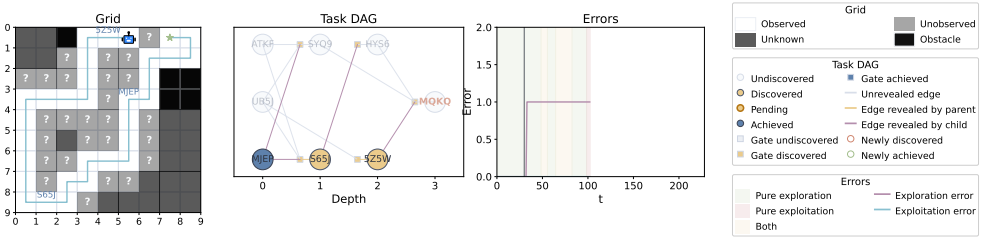
(a) Claude Opus 4.6, $t = 5$



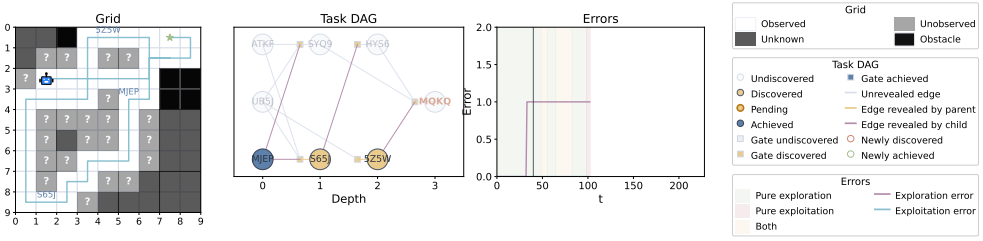
(b) Claude Opus 4.6, $t = 10$



(c) Claude Opus 4.6, $t = 20$

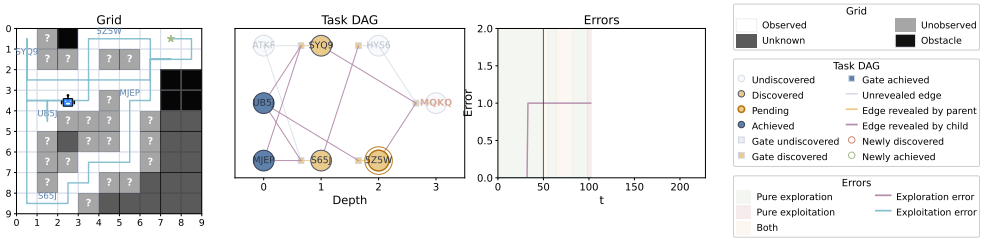


(d) Claude Opus 4.6, $t = 30$

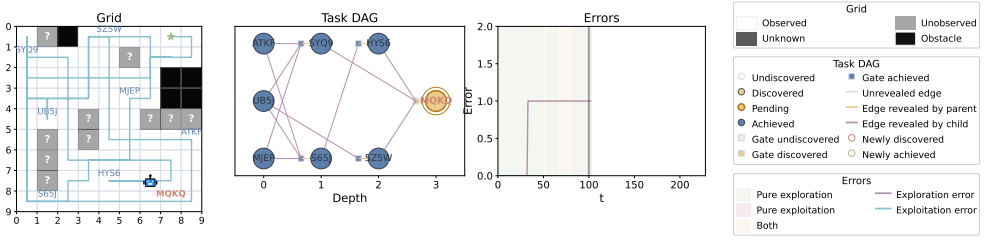


(e) Claude Opus 4.6, $t = 40$

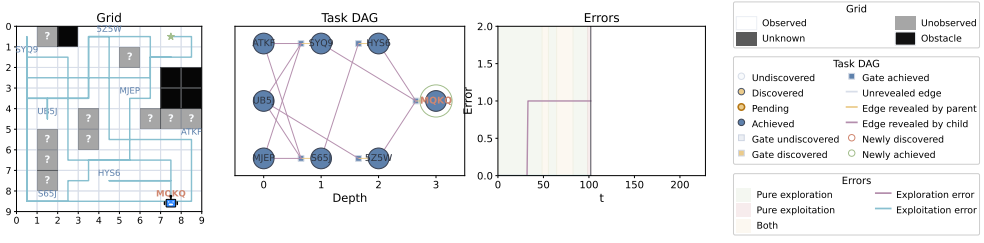
Figure 17: Results of action trajectories and metric values over each timestep for Claude Opus 4.6.



(a) Claude Opus 4.6, $t = 50$

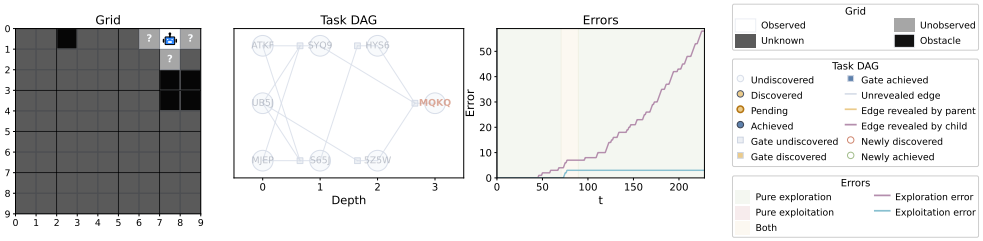


(b) Claude Opus 4.6, $t = 100$

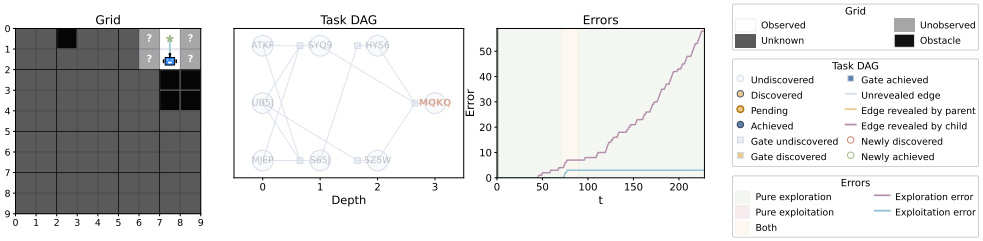


(c) Claude Opus 4.6, $t = 102$

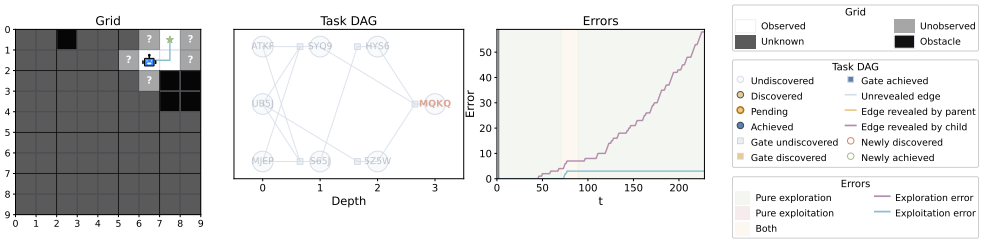
Figure 18: Results of action trajectories and metric values over each timestep for Claude Opus 4.6.



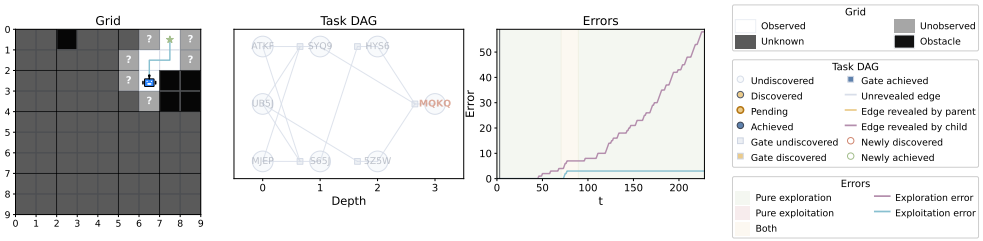
(a) Gemini 3 Flash, $t = 0$



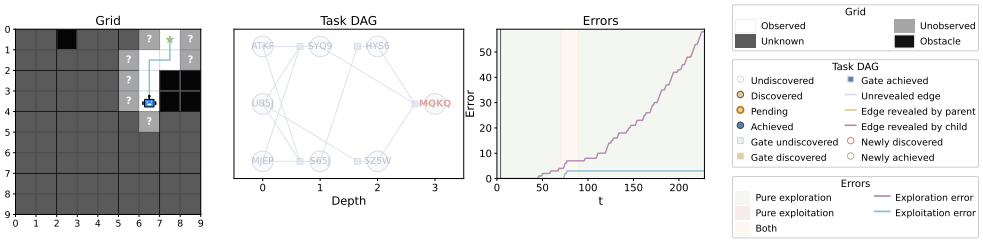
(b) Gemini 3 Flash, $t = 1$



(c) Gemini 3 Flash, $t = 2$

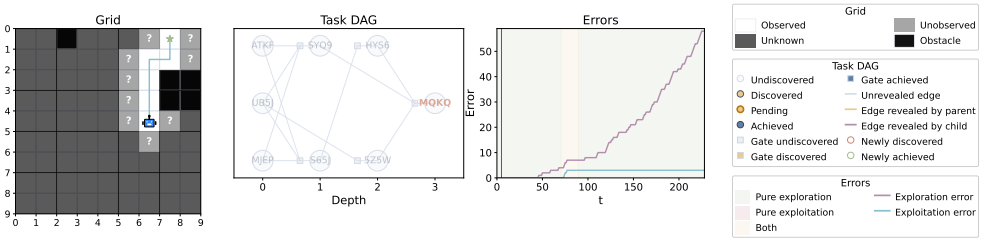


(d) Gemini 3 Flash, $t = 3$

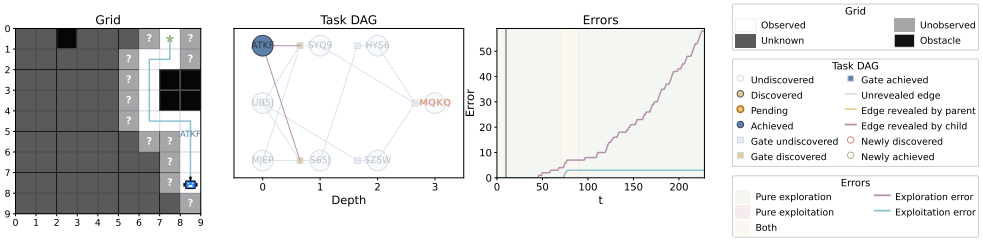


(e) Gemini 3 Flash, $t = 4$

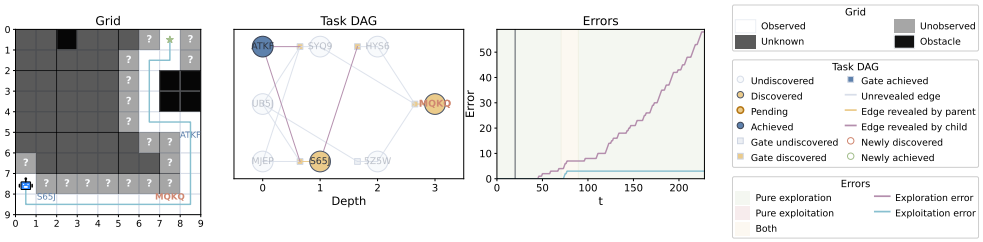
Figure 19: Results of action trajectories and metric values over each timestep for Gemini 3 Flash.



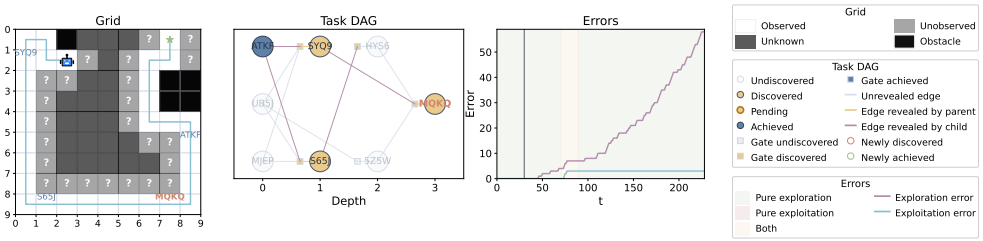
(a) Gemini 3 Flash, $t = 5$



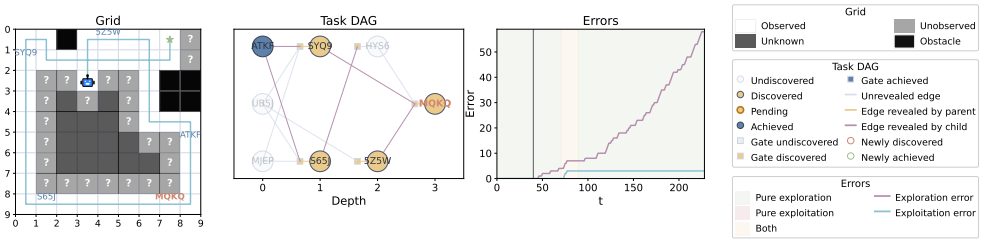
(b) Gemini 3 Flash, $t = 10$



(c) Gemini 3 Flash, $t = 20$

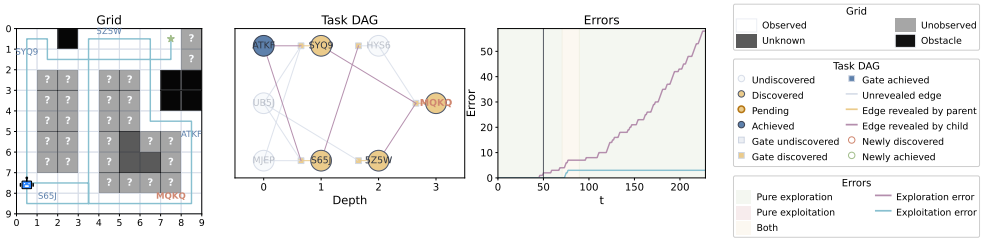


(d) Gemini 3 Flash, $t = 30$

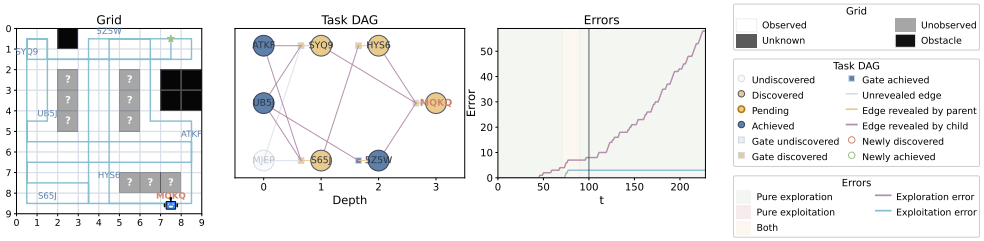


(e) Gemini 3 Flash, $t = 40$

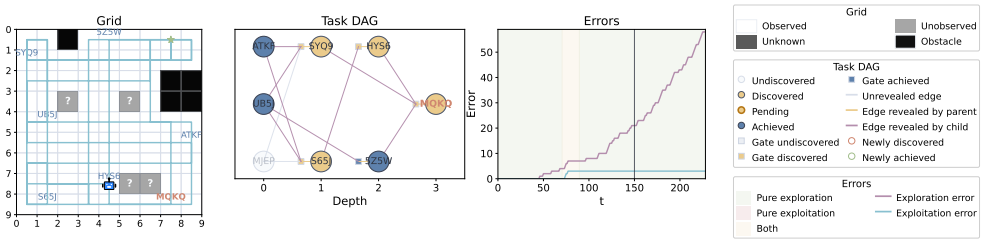
Figure 20: Results of action trajectories and metric values over each timestep for Gemini 3 Flash.



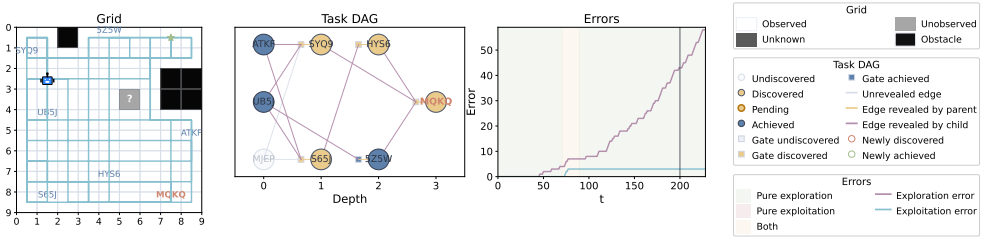
(a) Gemini 3 Flash, $t = 50$



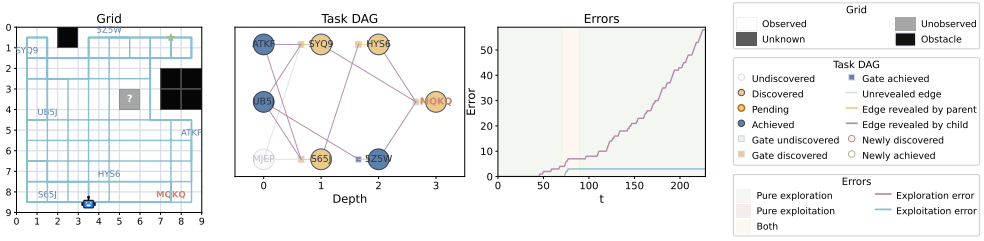
(b) Gemini 3 Flash, $t = 100$



(c) Gemini 3 Flash, $t = 150$

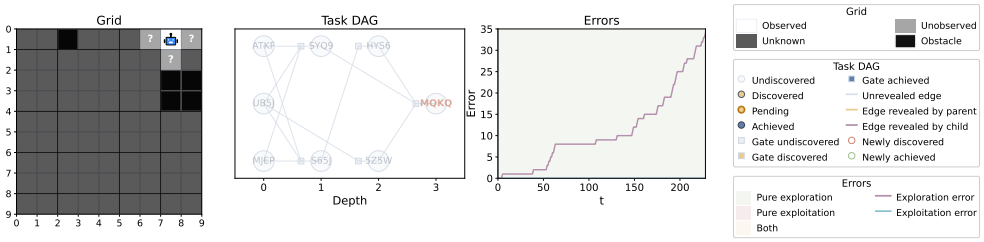


(d) Gemini 3 Flash, $t = 200$

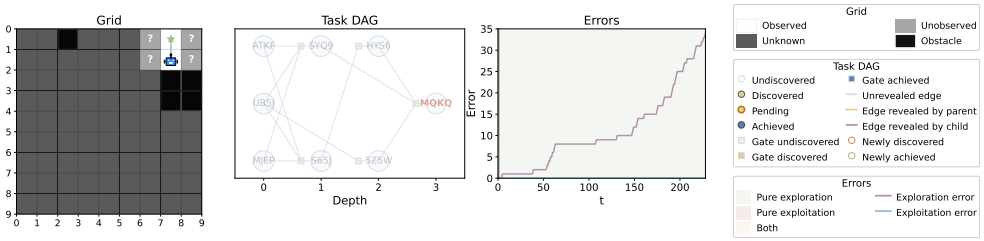


(e) Gemini 3 Flash, $t = 228$

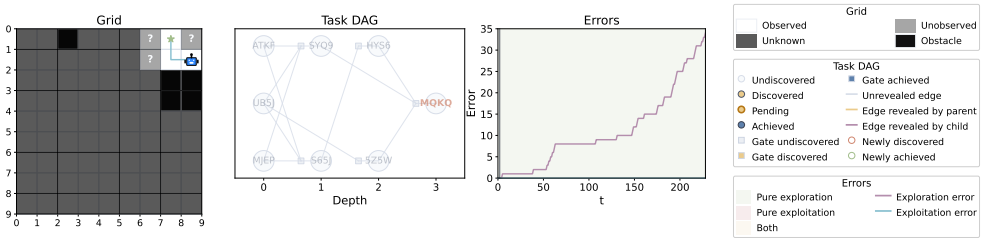
Figure 21: Results of action trajectories and metric values over each timestep for Gemini 3 Flash.



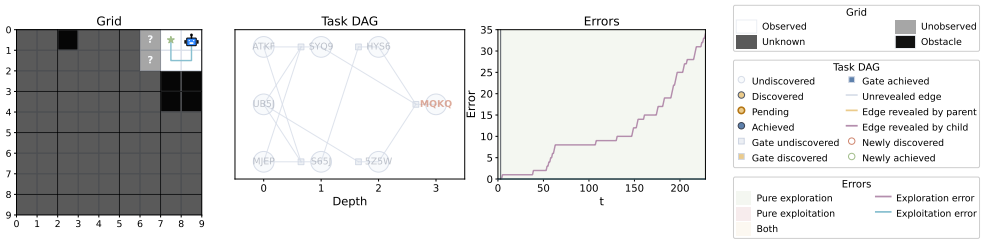
(a) Gemini 3.1 Flash Lite, $t = 0$



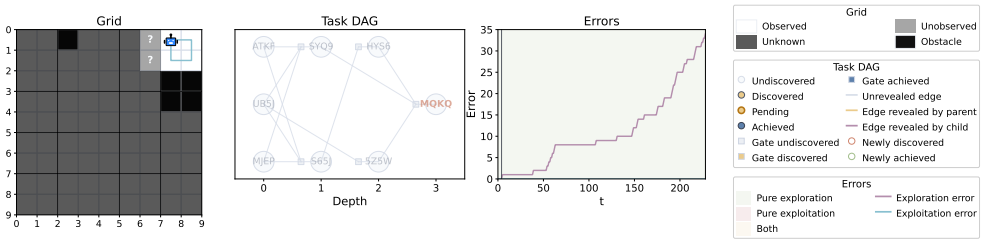
(b) Gemini 3.1 Flash Lite, $t = 1$



(c) Gemini 3.1 Flash Lite, $t = 2$

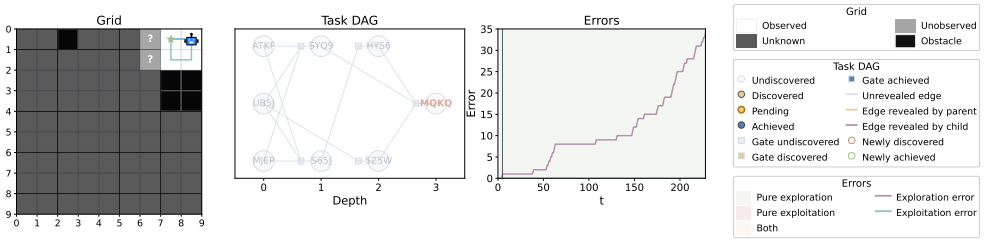


(d) Gemini 3.1 Flash Lite, $t = 3$

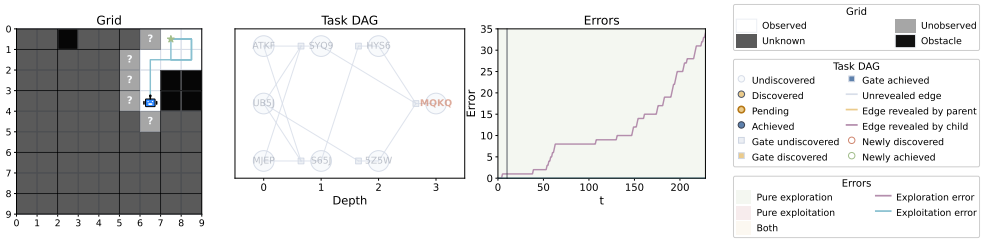


(e) Gemini 3.1 Flash Lite, $t = 4$

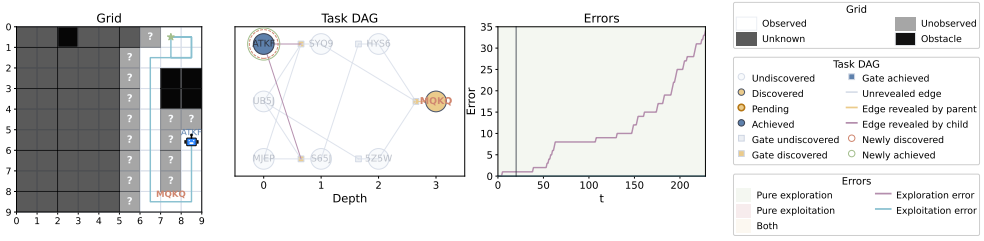
Figure 22: Results of action trajectories and metric values over each timestep for Gemini 3.1 Flash Lite.



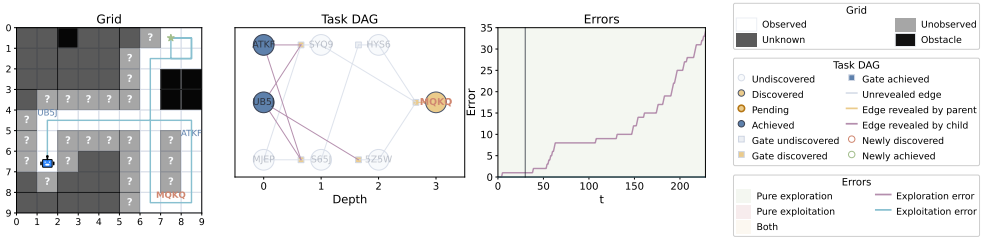
(a) Gemini 3.1 Flash Lite, $t = 5$



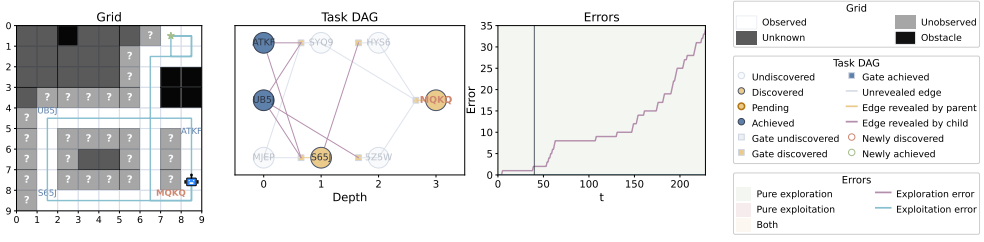
(b) Gemini 3.1 Flash Lite, $t = 10$



(c) Gemini 3.1 Flash Lite, $t = 20$

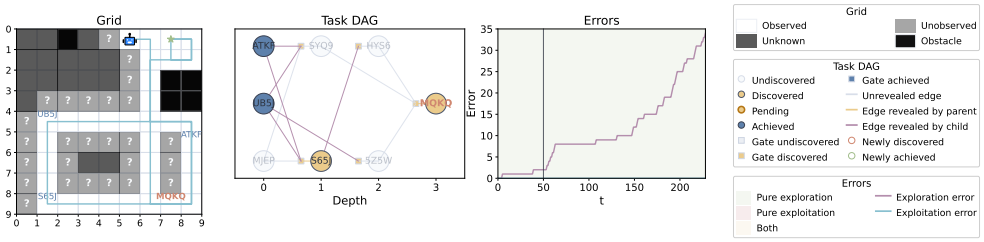


(d) Gemini 3.1 Flash Lite, $t = 30$

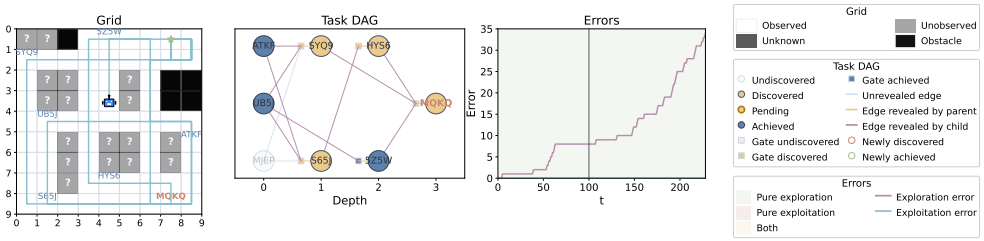


(e) Gemini 3.1 Flash Lite, $t = 40$

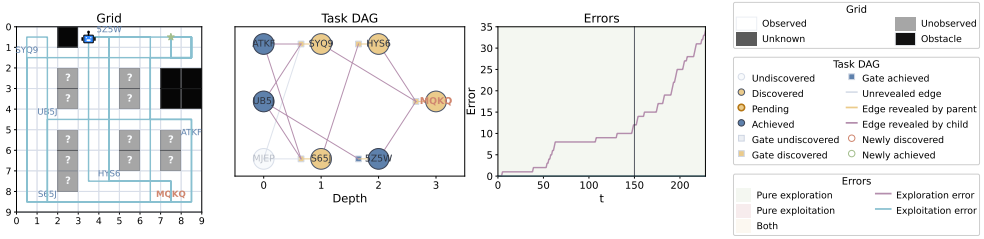
Figure 23: Results of action trajectories and metric values over each timestep for Gemini 3.1 Flash Lite.



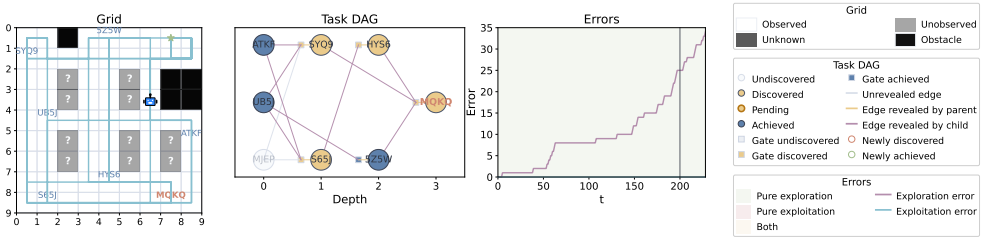
(a) Gemini 3.1 Flash Lite, $t = 50$



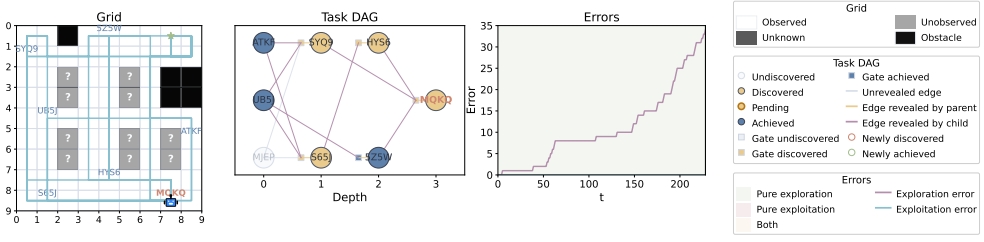
(b) Gemini 3.1 Flash Lite, $t = 100$



(c) Gemini 3.1 Flash Lite, $t = 150$

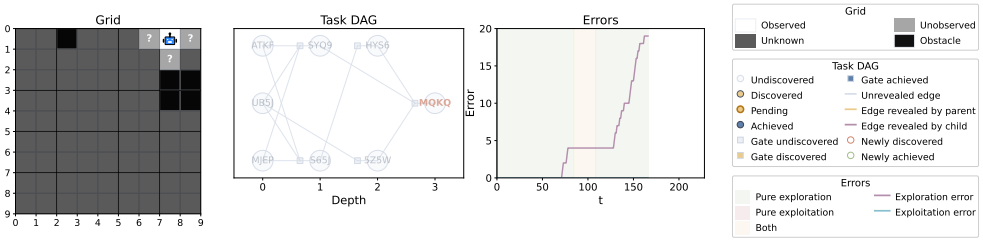


(d) Gemini 3.1 Flash Lite, $t = 200$

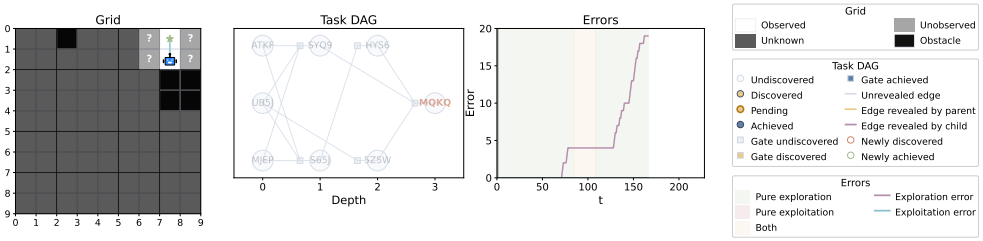


(e) Gemini 3.1 Flash Lite, $t = 228$

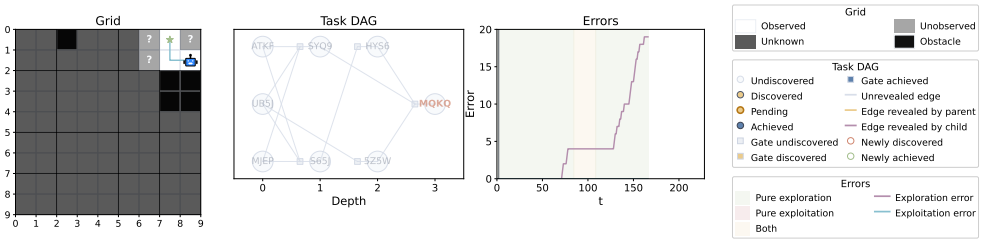
Figure 24: Results of action trajectories and metric values over each timestep for Gemini 3.1 Flash Lite.



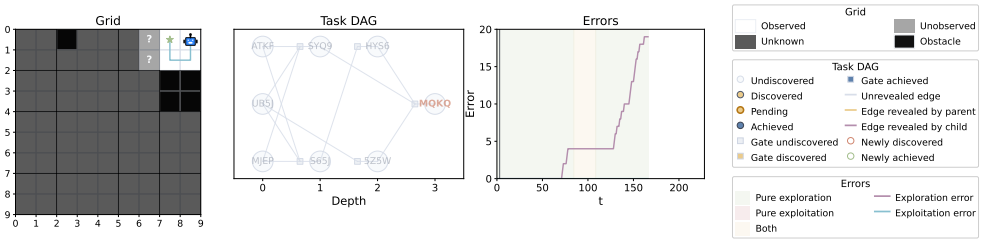
(a) Gemini 3.1 Pro, $t = 0$



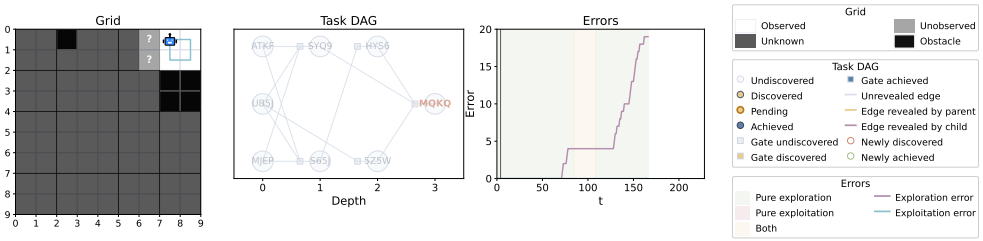
(b) Gemini 3.1 Pro, $t = 1$



(c) Gemini 3.1 Pro, $t = 2$

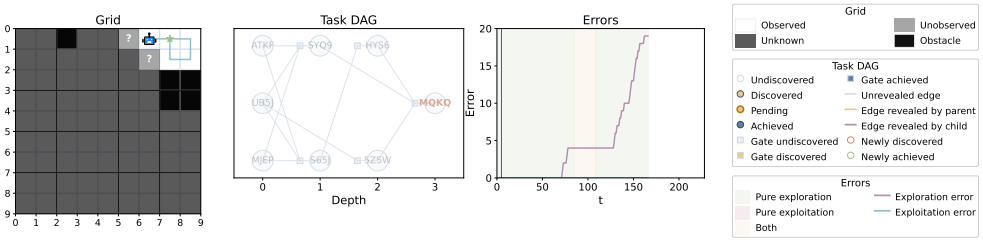


(d) Gemini 3.1 Pro, $t = 3$

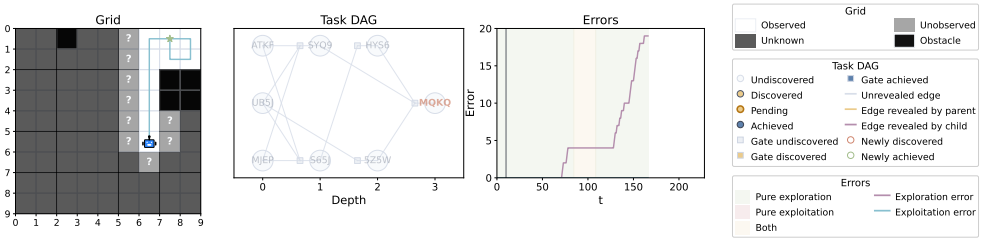


(e) Gemini 3.1 Pro, $t = 4$

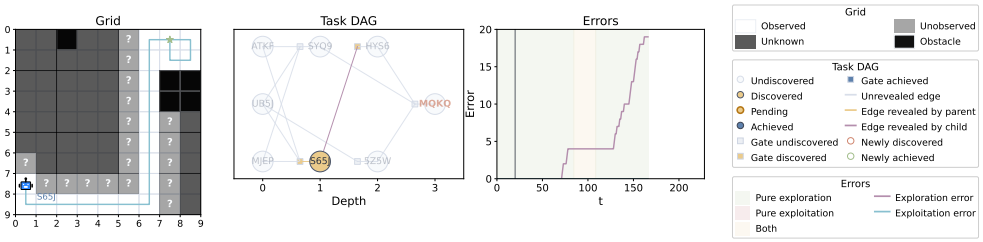
Figure 25: Results of action trajectories and metric values over each timestep for Gemini 3.1 Pro.



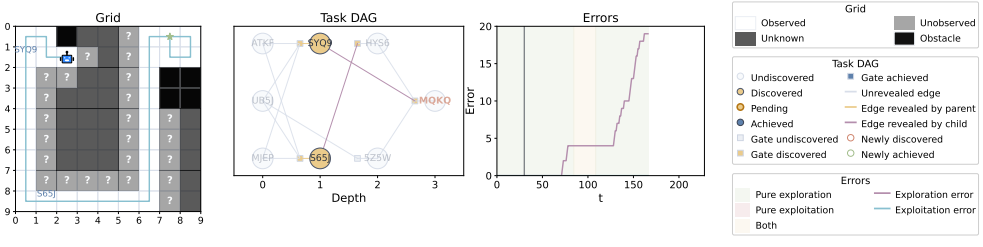
(a) Gemini 3.1 Pro, $t = 5$



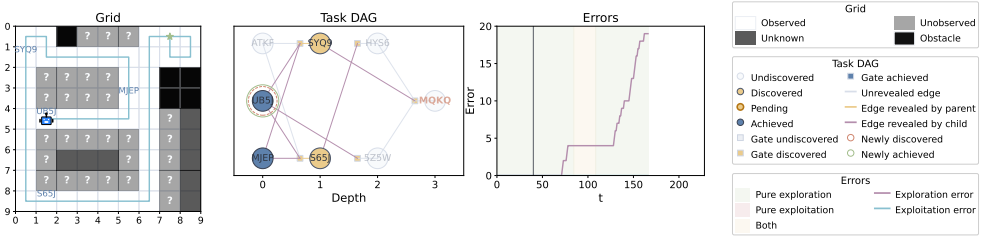
(b) Gemini 3.1 Pro, $t = 10$



(c) Gemini 3.1 Pro, $t = 20$

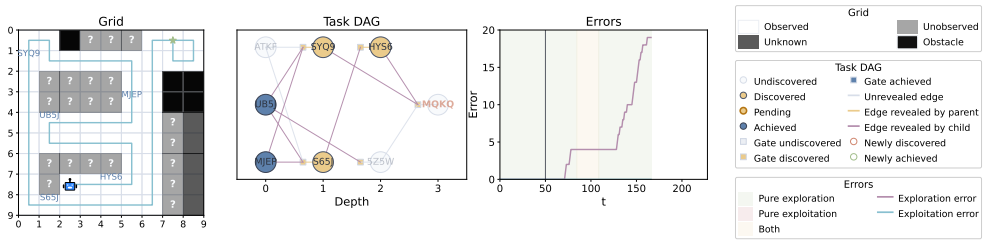


(d) Gemini 3.1 Pro, $t = 30$

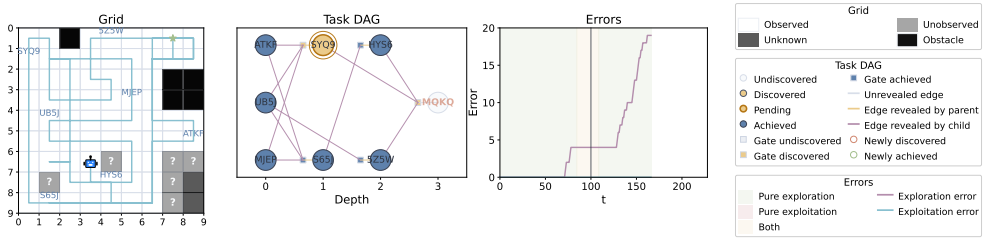


(e) Gemini 3.1 Pro, $t = 40$

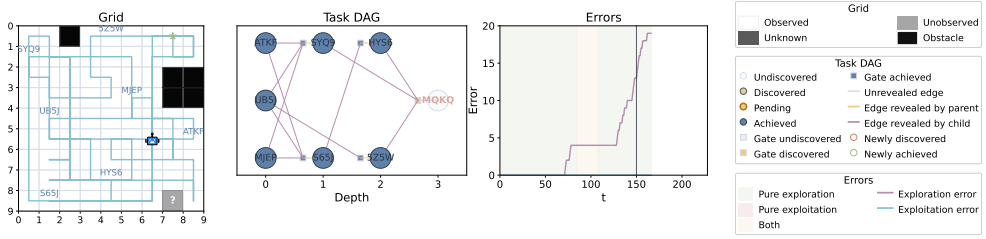
Figure 26: Results of action trajectories and metric values over each timestep for Gemini 3.1 Pro.



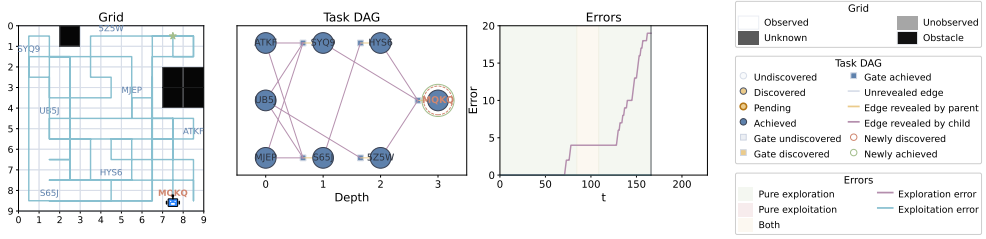
(a) Gemini 3.1 Pro, $t = 50$



(b) Gemini 3.1 Pro, $t = 100$

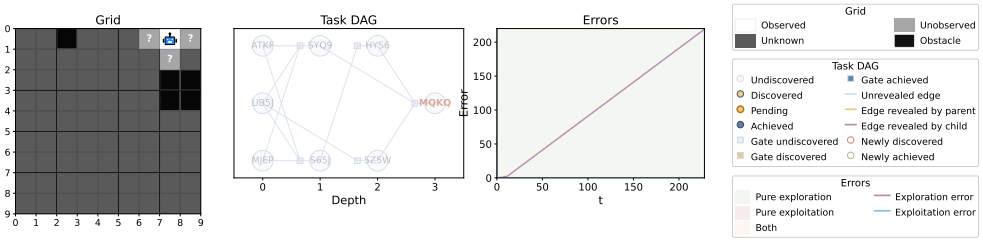


(c) Gemini 3.1 Pro, $t = 150$

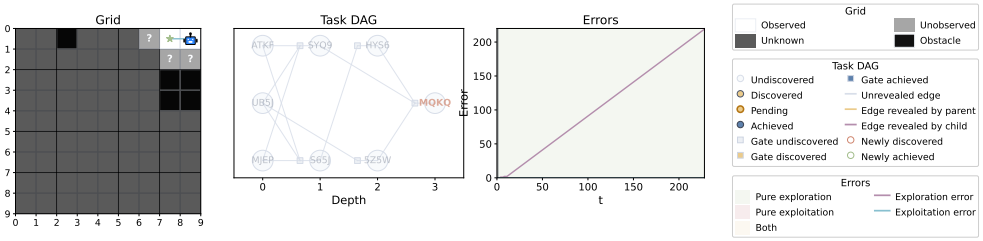


(d) Gemini 3.1 Pro, $t = 166$

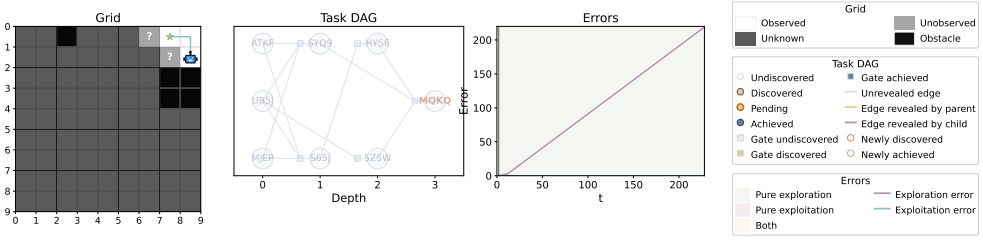
Figure 27: Results of action trajectories and metric values over each timestep for Gemini 3.1 Pro.



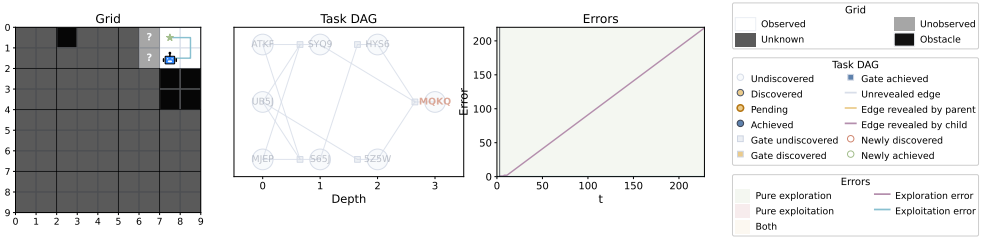
(a) GPT-4.1 mini, $t = 0$



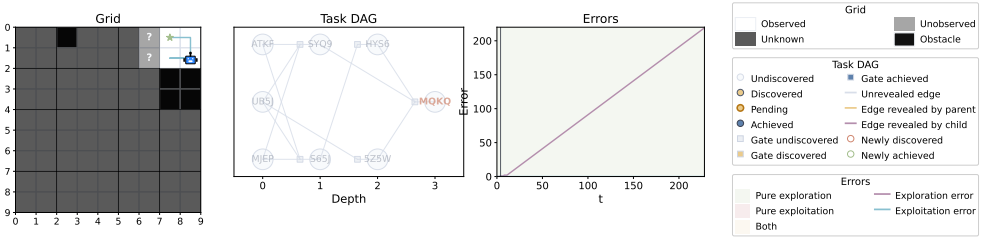
(b) GPT-4.1 mini, $t = 1$



(c) GPT-4.1 mini, $t = 2$

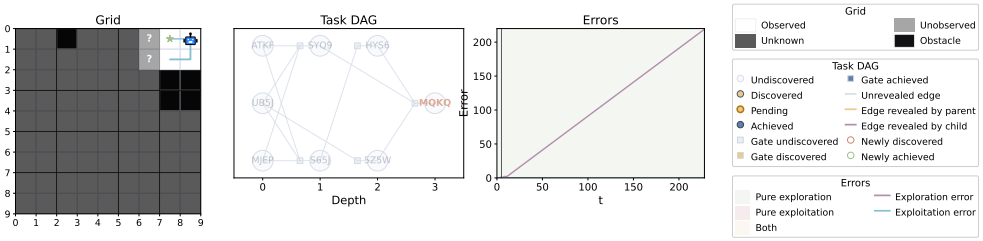


(d) GPT-4.1 mini, $t = 3$

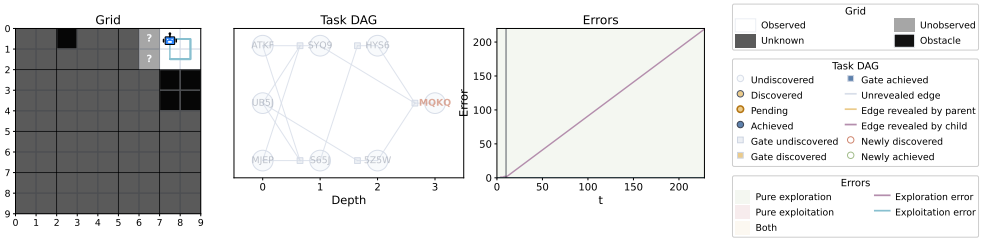


(e) GPT-4.1 mini, $t = 4$

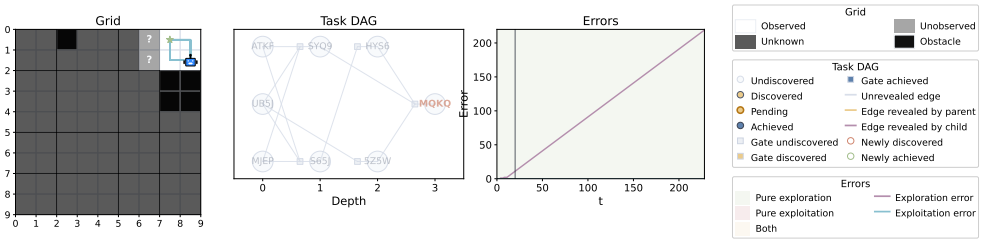
Figure 28: Results of action trajectories and metric values over each timestep for GPT-4.1 mini.



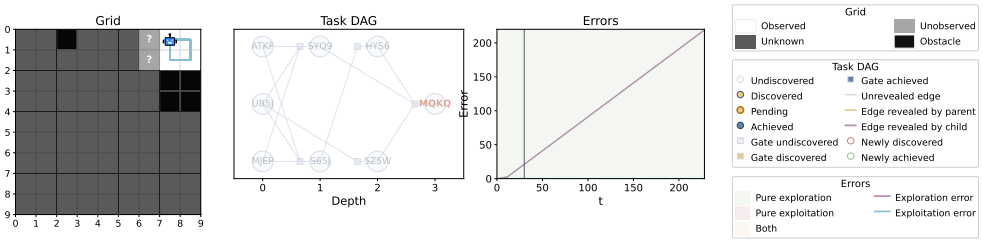
(a) GPT-4.1 mini, $t = 5$



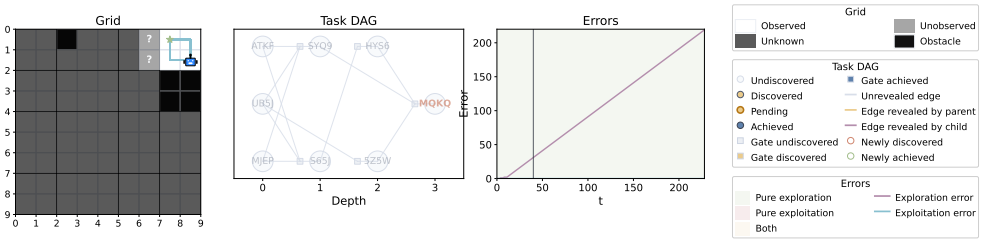
(b) GPT-4.1 mini, $t = 10$



(c) GPT-4.1 mini, $t = 20$

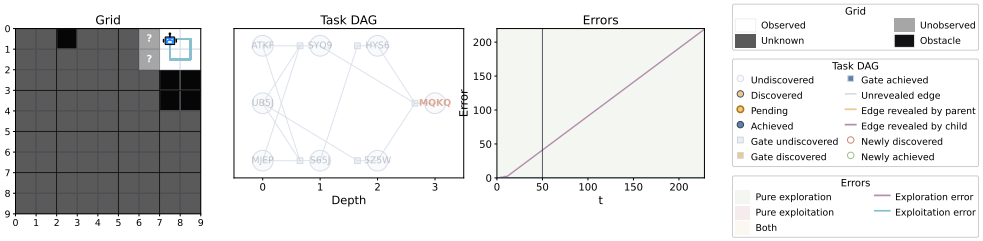


(d) GPT-4.1 mini, $t = 30$

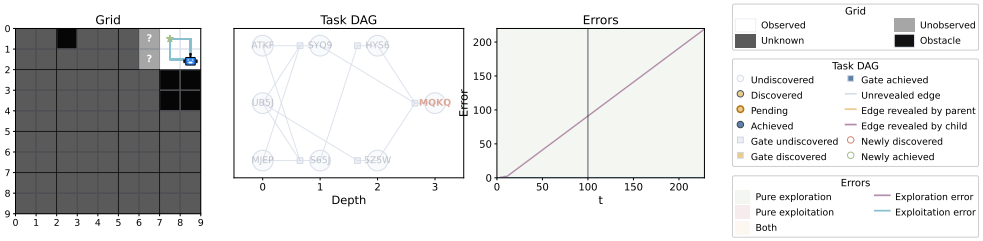


(e) GPT-4.1 mini, $t = 40$

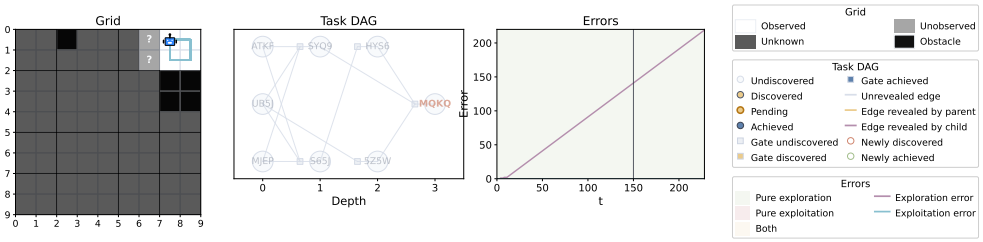
Figure 29: Results of action trajectories and metric values over each timestep for GPT-4.1 mini.



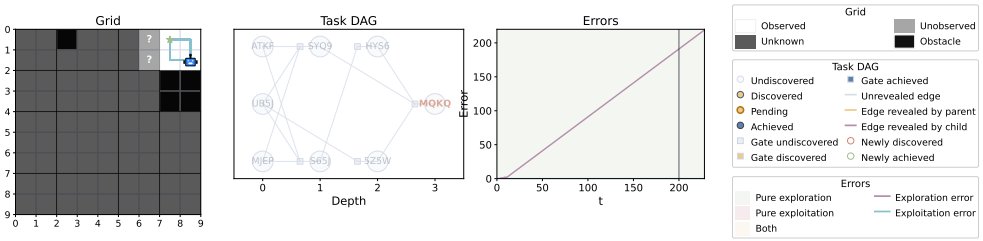
(a) GPT-4.1 mini, $t = 50$



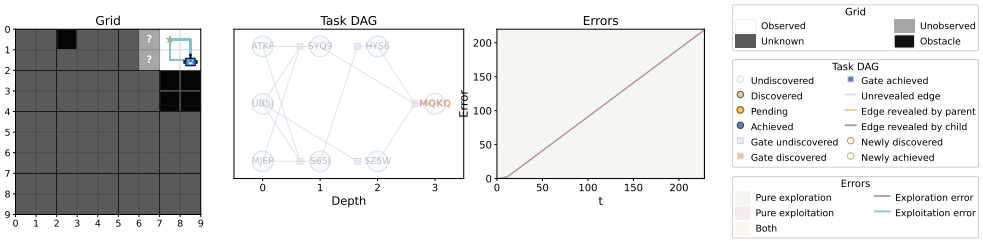
(b) GPT-4.1 mini, $t = 100$



(c) GPT-4.1 mini, $t = 150$

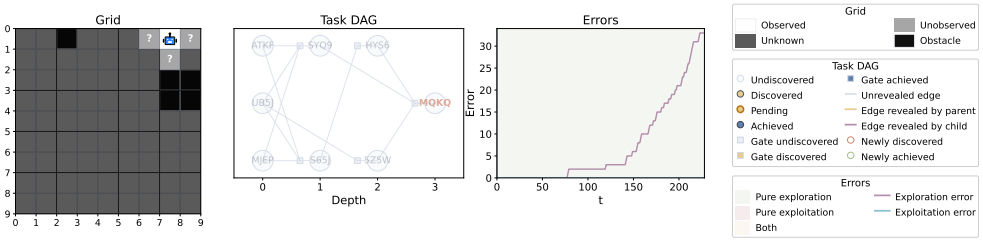


(d) GPT-4.1 mini, $t = 200$

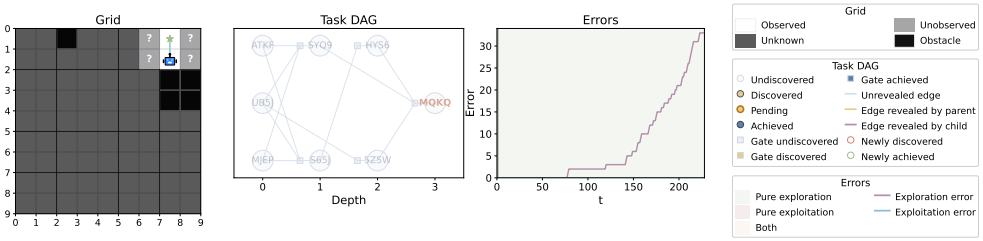


(e) GPT-4.1 mini, $t = 228$

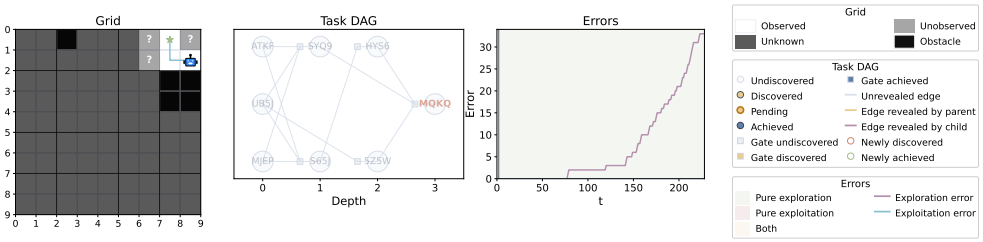
Figure 30: Results of action trajectories and metric values over each timestep for GPT-4.1 mini.



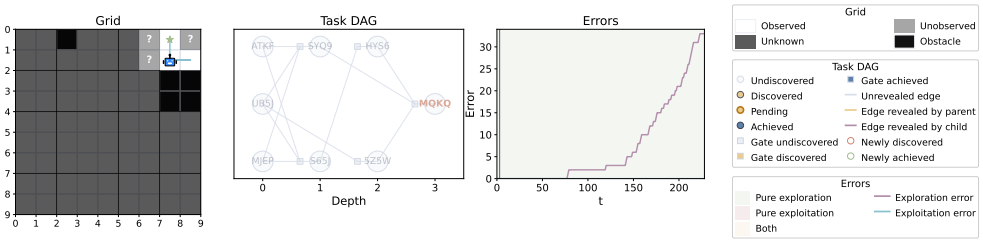
(a) GPT-4.1, $t = 0$



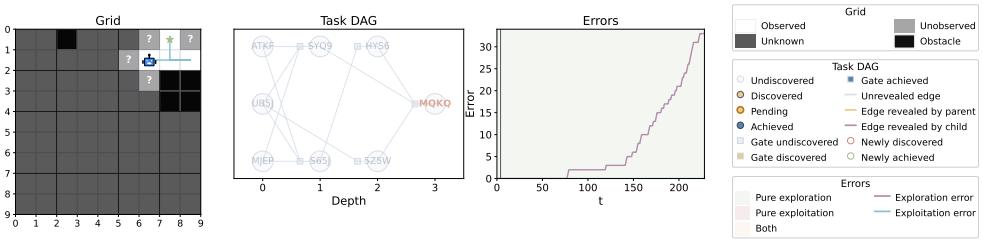
(b) GPT-4.1, $t = 1$



(c) GPT-4.1, $t = 2$

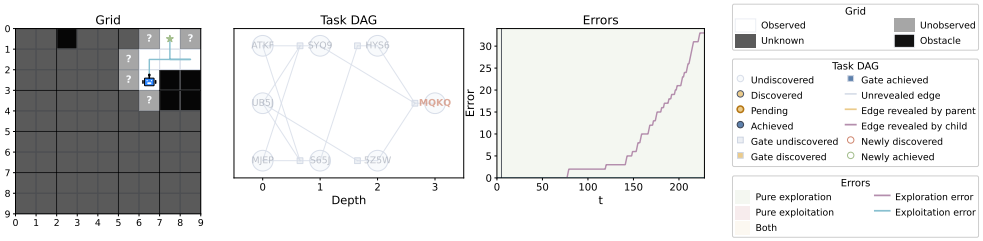


(d) GPT-4.1, $t = 3$

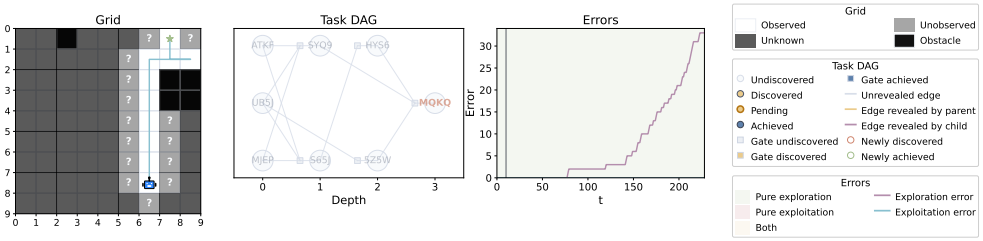


(e) GPT-4.1, $t = 4$

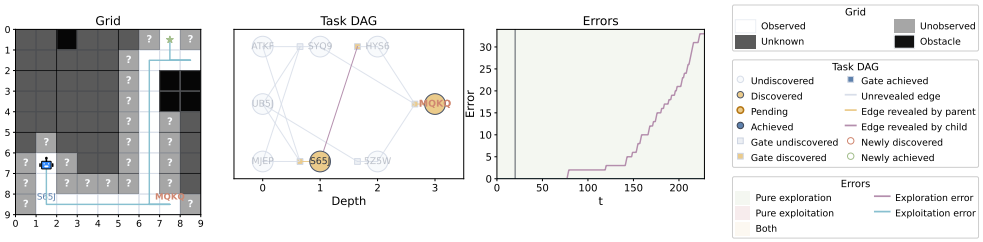
Figure 31: Results of action trajectories and metric values over each timestep for GPT-4.1.



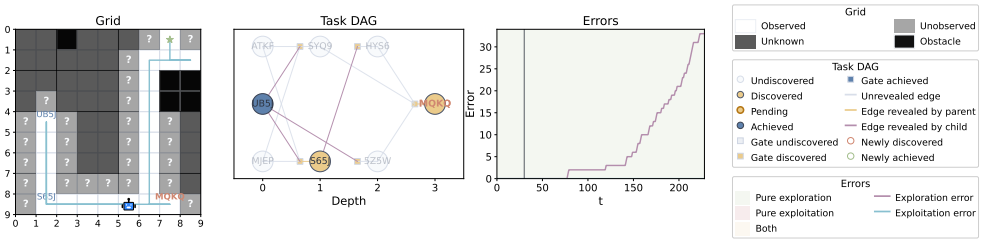
(a) GPT-4.1, $t = 5$



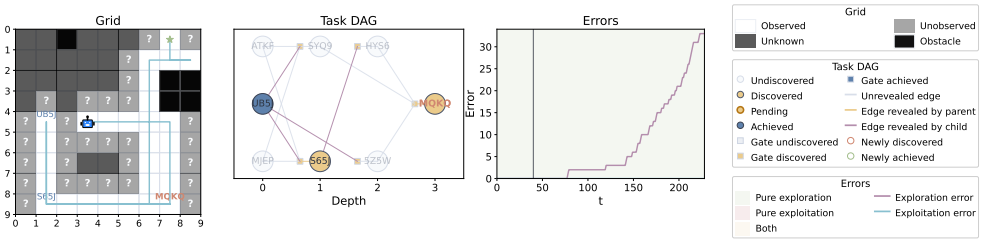
(b) GPT-4.1, $t = 10$



(c) GPT-4.1, $t = 20$

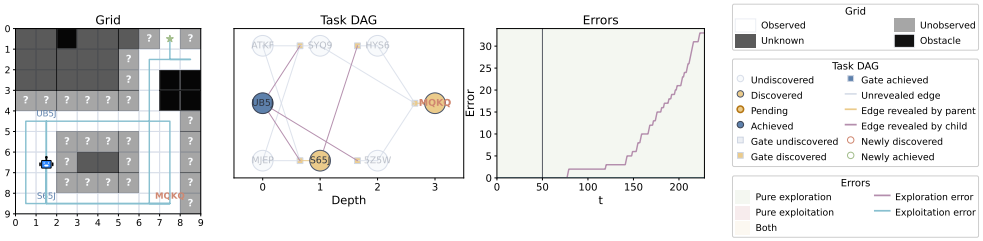


(d) GPT-4.1, $t = 30$

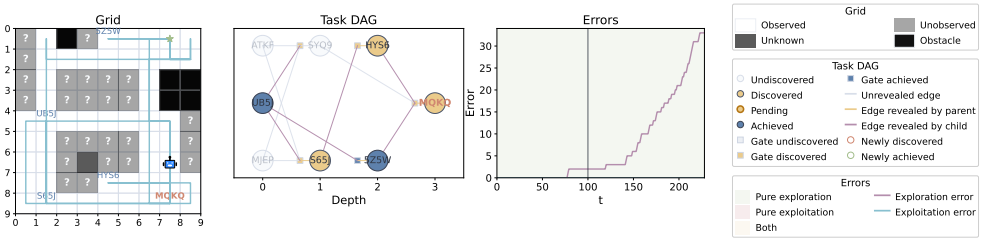


(e) GPT-4.1, $t = 40$

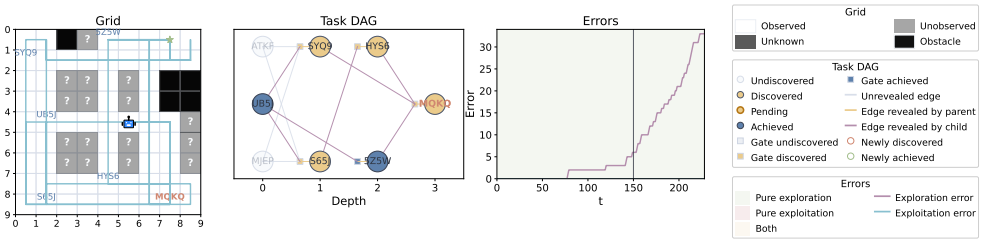
Figure 32: Results of action trajectories and metric values over each timestep for GPT-4.1.



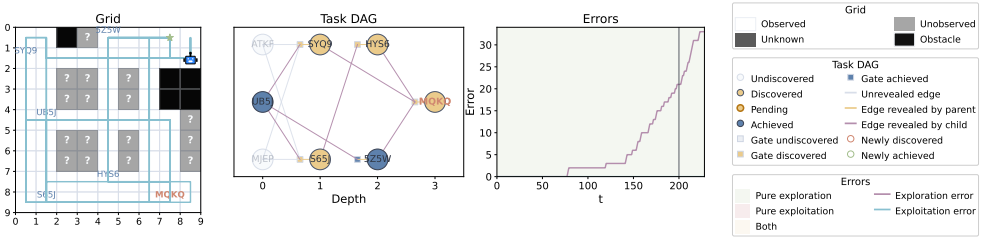
(a) GPT-4.1, $t = 50$



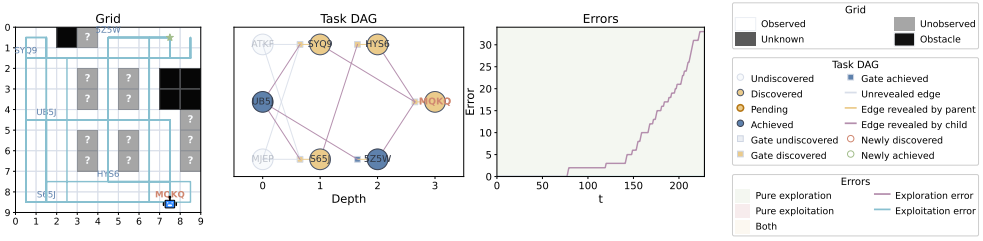
(b) GPT-4.1, $t = 100$



(c) GPT-4.1, $t = 150$

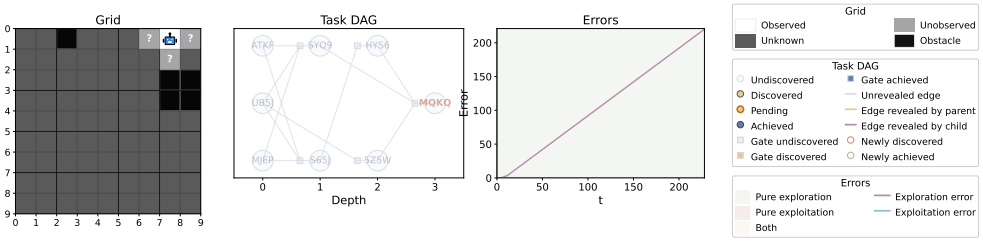


(d) GPT-4.1, $t = 200$

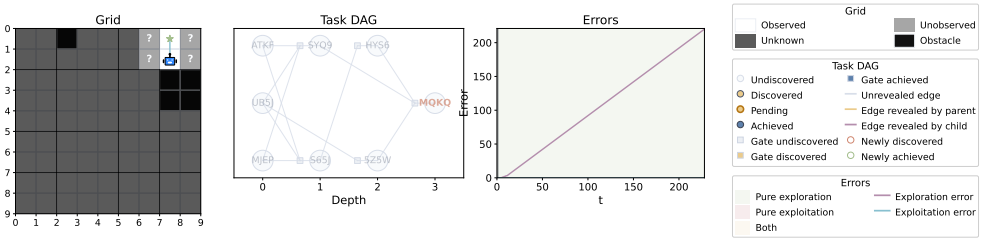


(e) GPT-4.1, $t = 228$

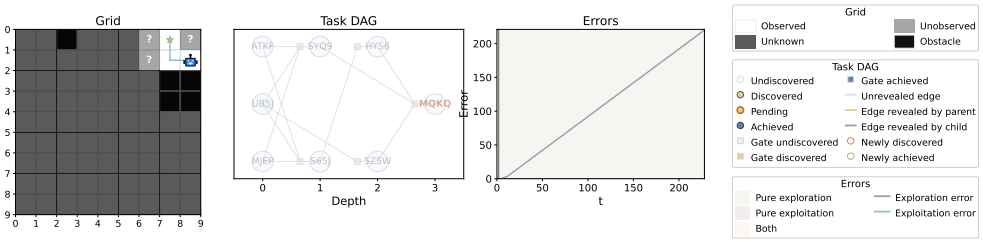
Figure 33: Results of action trajectories and metric values over each timestep for GPT-4.1.



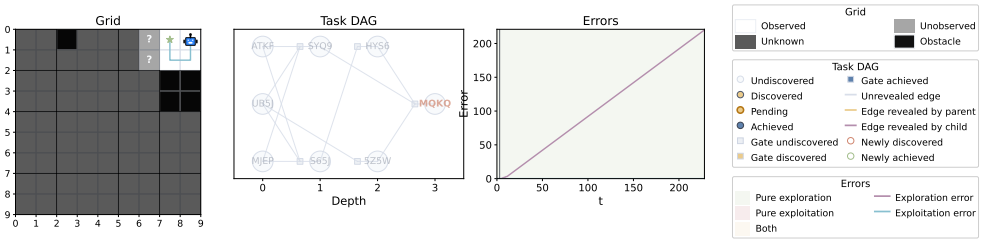
(a) GPT-5.4 mini, $t = 0$



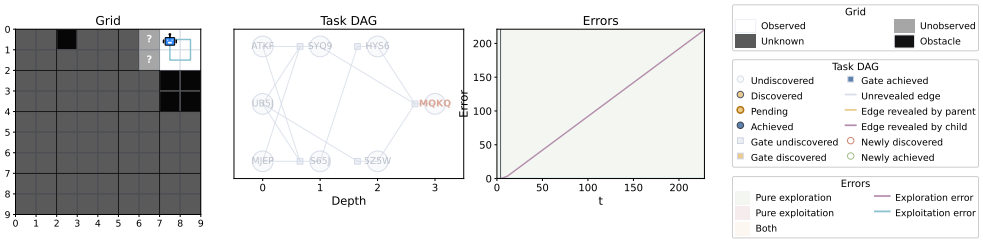
(b) GPT-5.4 mini, $t = 1$



(c) GPT-5.4 mini, $t = 2$

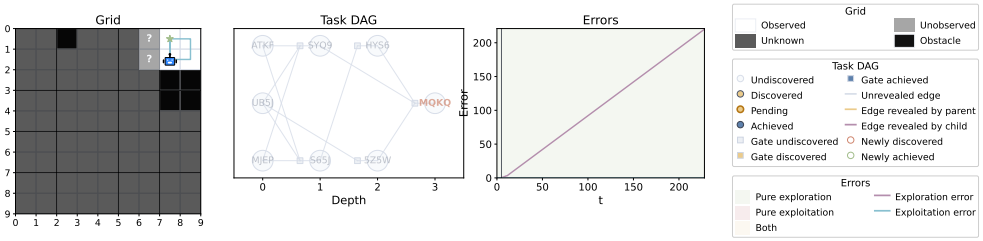


(d) GPT-5.4 mini, $t = 3$

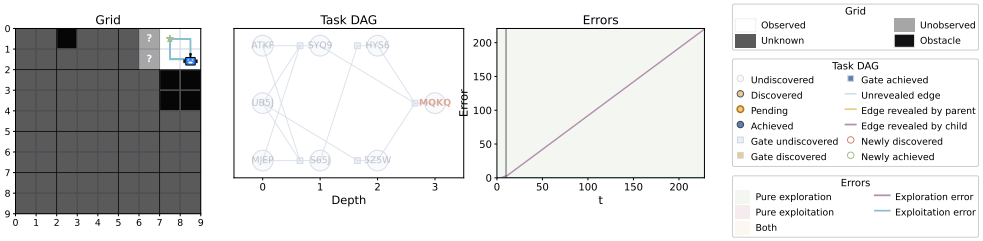


(e) GPT-5.4 mini, $t = 4$

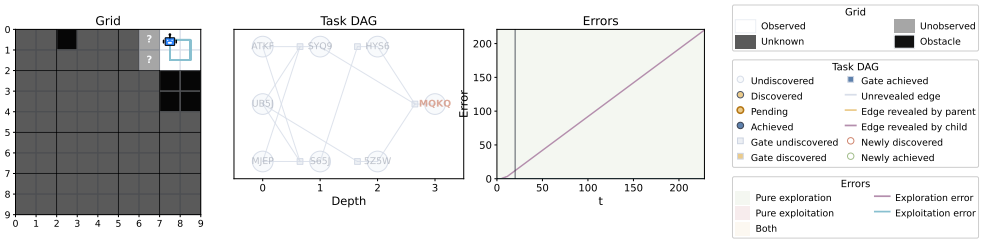
Figure 34: Results of action trajectories and metric values over each timestep for GPT-5.4 mini.



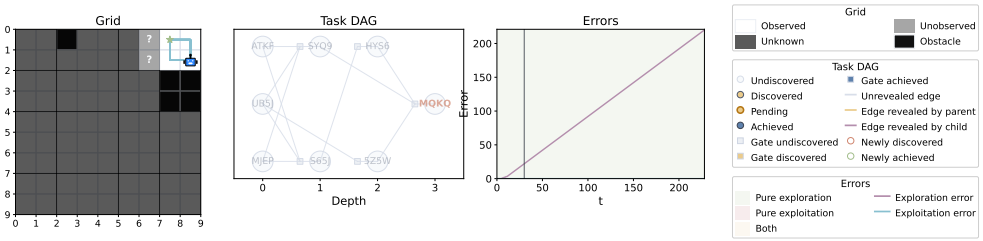
(a) GPT-5.4 mini, $t = 5$



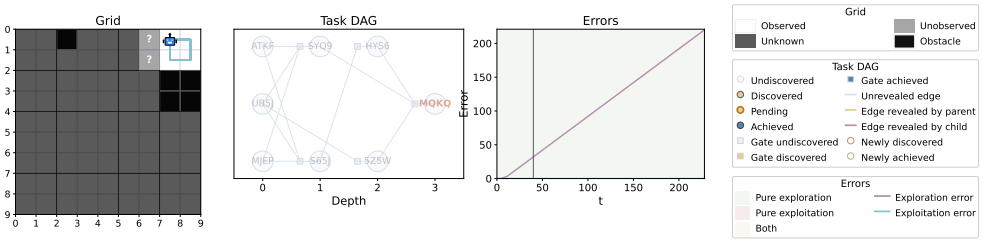
(b) GPT-5.4 mini, $t = 10$



(c) GPT-5.4 mini, $t = 20$

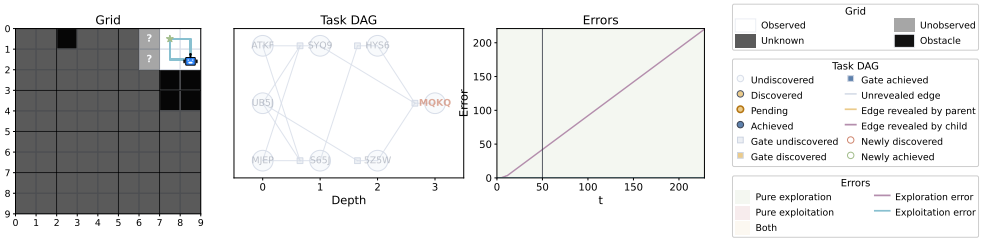


(d) GPT-5.4 mini, $t = 30$

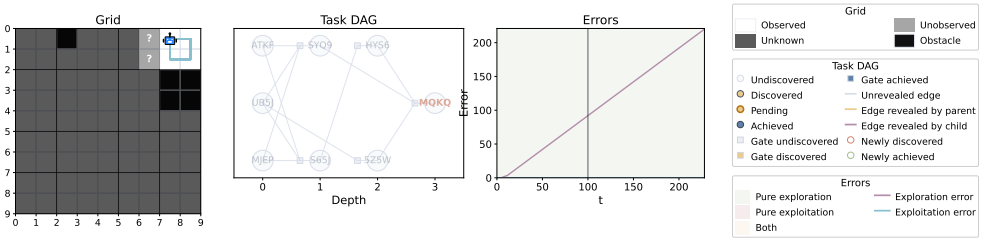


(e) GPT-5.4 mini, $t = 40$

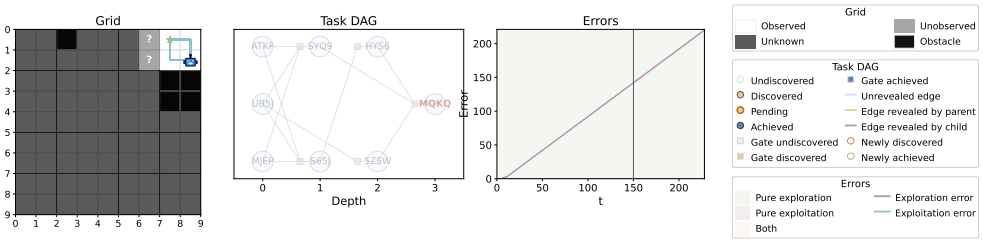
Figure 35: Results of action trajectories and metric values over each timestep for GPT-5.4 mini.



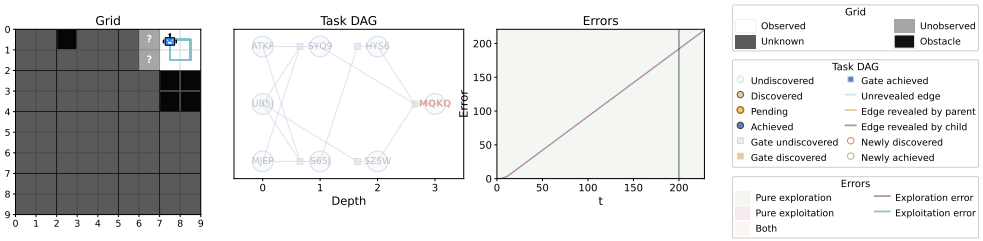
(a) GPT-5.4 mini, $t = 50$



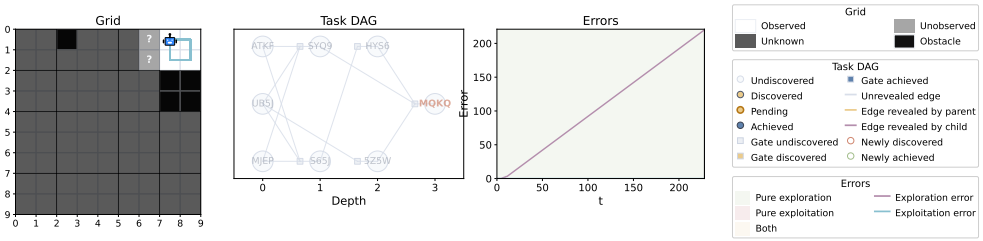
(b) GPT-5.4 mini, $t = 100$



(c) GPT-5.4 mini, $t = 150$

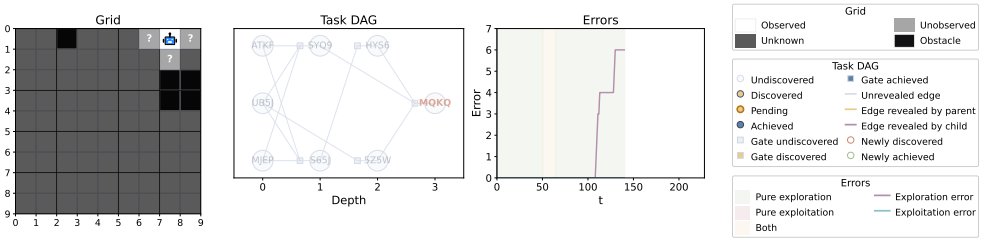


(d) GPT-5.4 mini, $t = 200$

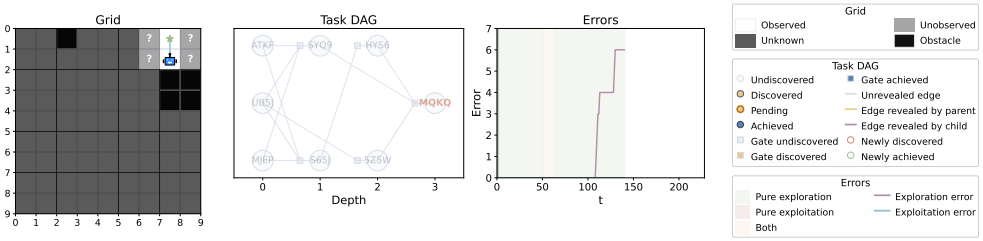


(e) GPT-5.4 mini, $t = 228$

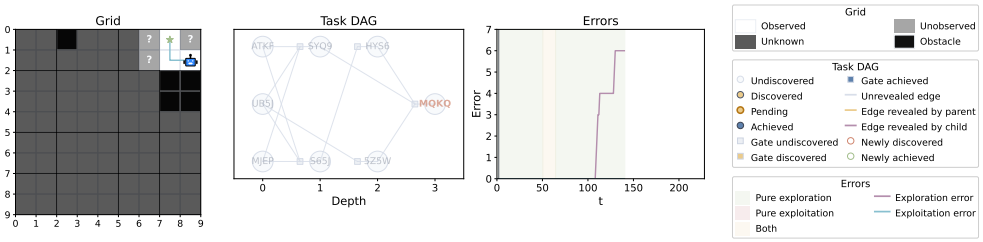
Figure 36: Results of action trajectories and metric values over each timestep for GPT-5.4 mini.



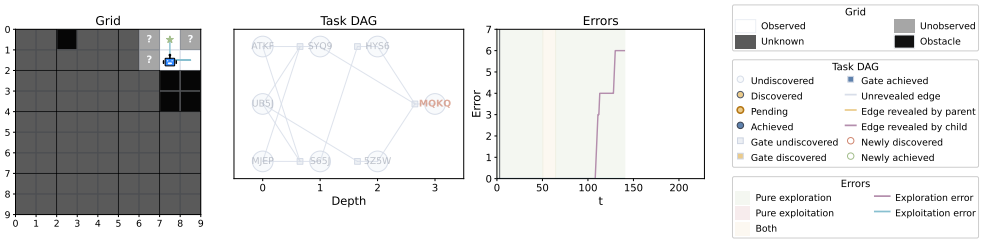
(a) GPT-5.4, $t = 0$



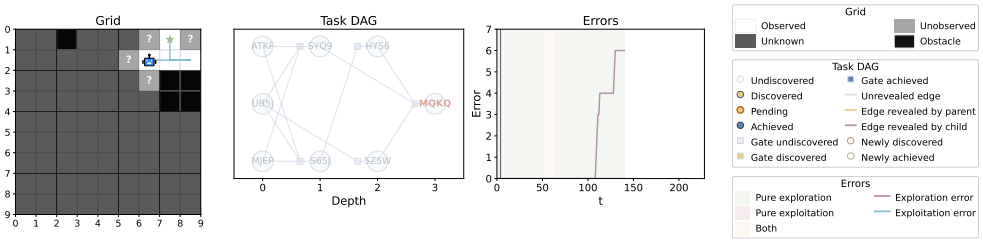
(b) GPT-5.4, $t = 1$



(c) GPT-5.4, $t = 2$

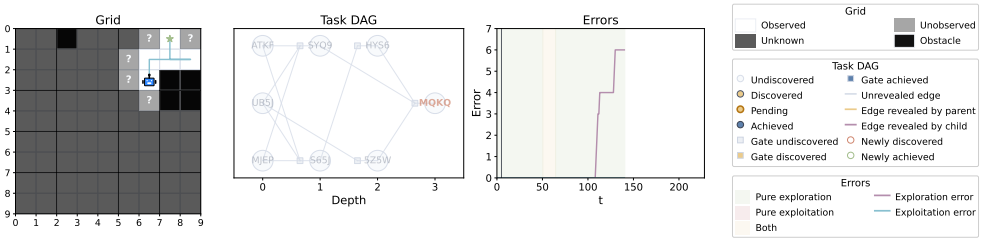


(d) GPT-5.4, $t = 3$

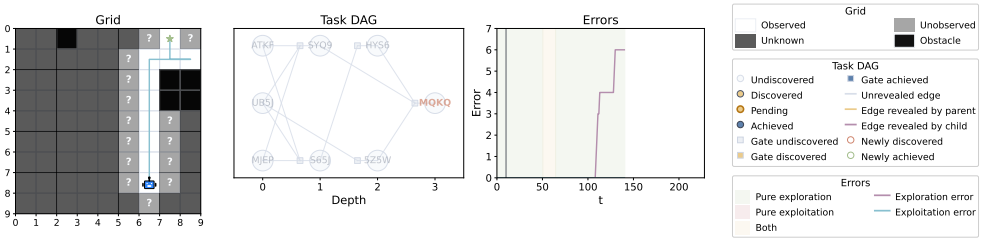


(e) GPT-5.4, $t = 4$

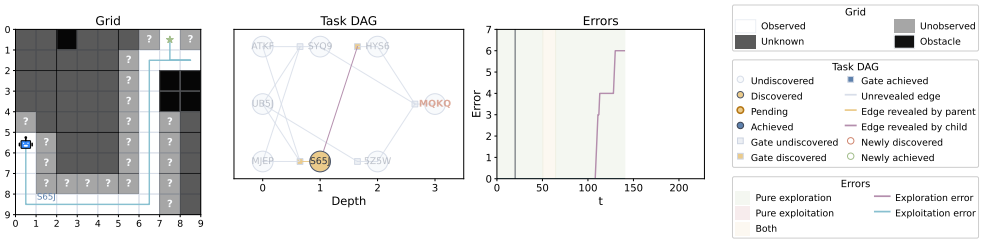
Figure 37: Results of action trajectories and metric values over each timestep for GPT-5.4.



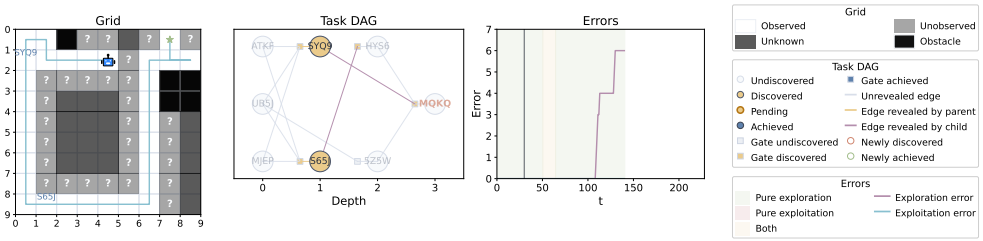
(a) GPT-5.4, $t = 5$



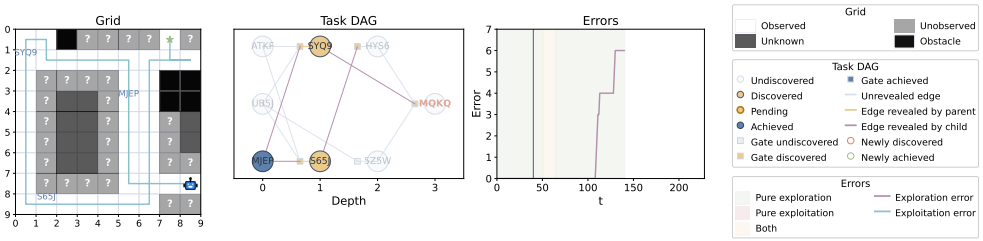
(b) GPT-5.4, $t = 10$



(c) GPT-5.4, $t = 20$

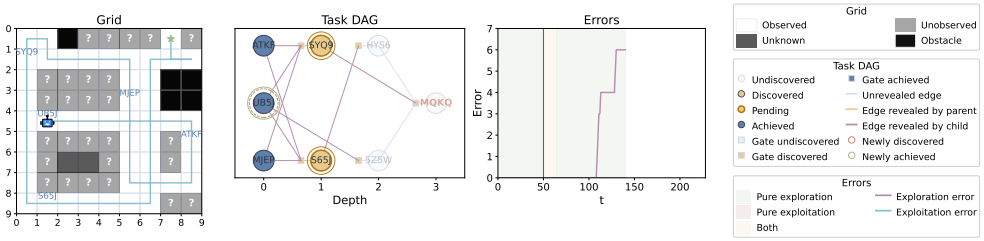


(d) GPT-5.4, $t = 30$

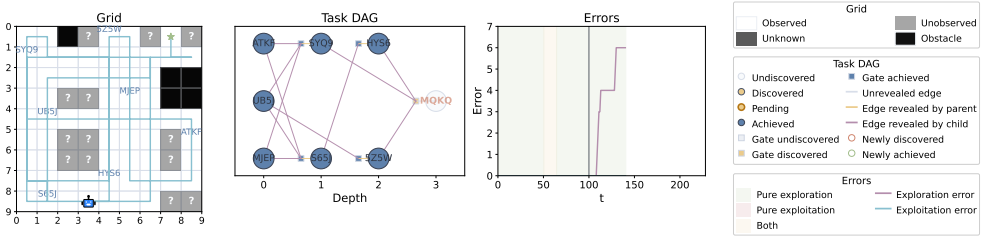


(e) GPT-5.4, $t = 40$

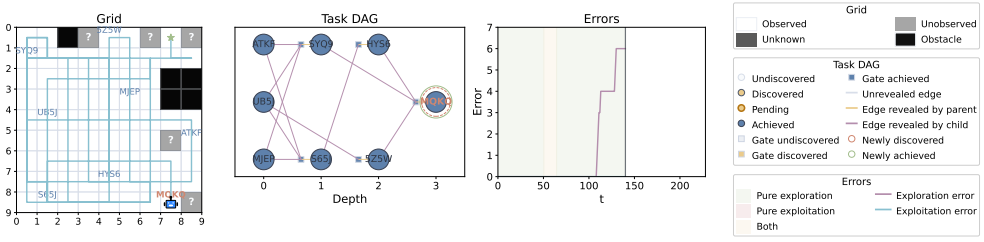
Figure 38: Results of action trajectories and metric values over each timestep for GPT-5.4.



(a) GPT-5.4, $t = 50$

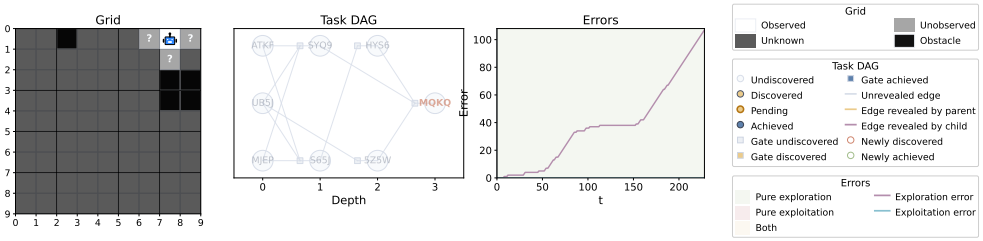


(b) GPT-5.4, $t = 100$

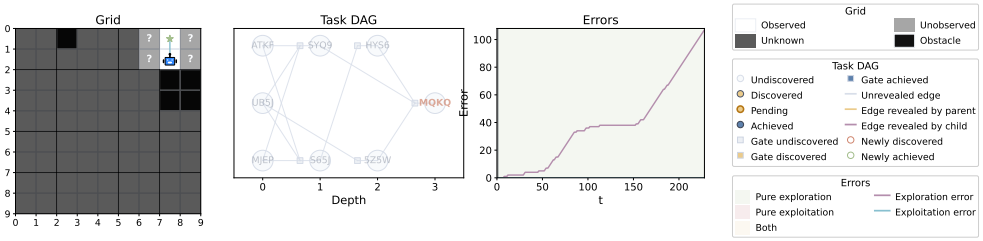


(c) GPT-5.4, $t = 140$

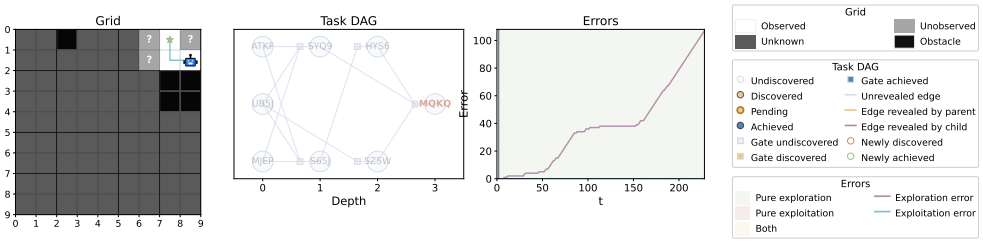
Figure 39: Results of action trajectories and metric values over each timestep for GPT-5.4.



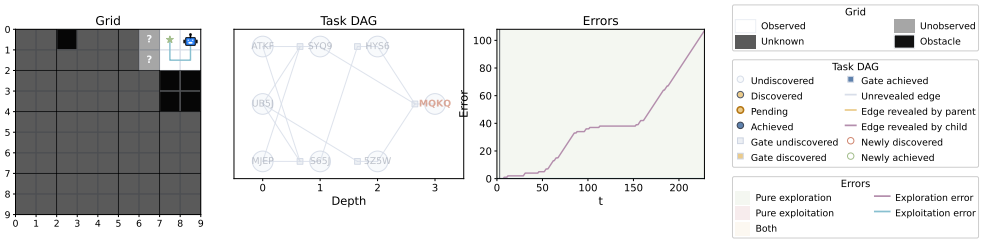
(a) GPT-OSS-120B, $t = 0$



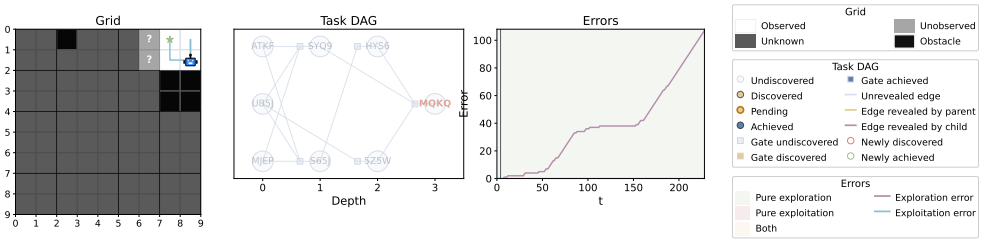
(b) GPT-OSS-120B, $t = 1$



(c) GPT-OSS-120B, $t = 2$

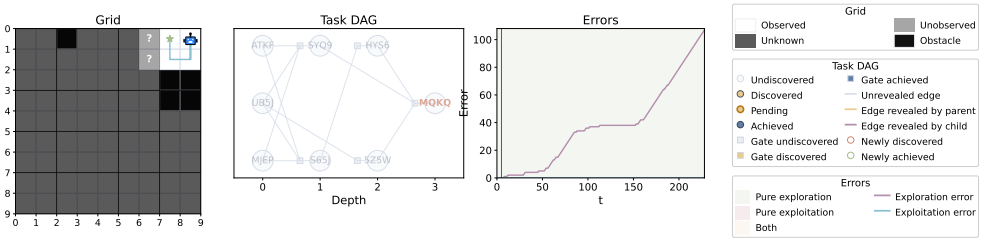


(d) GPT-OSS-120B, $t = 3$

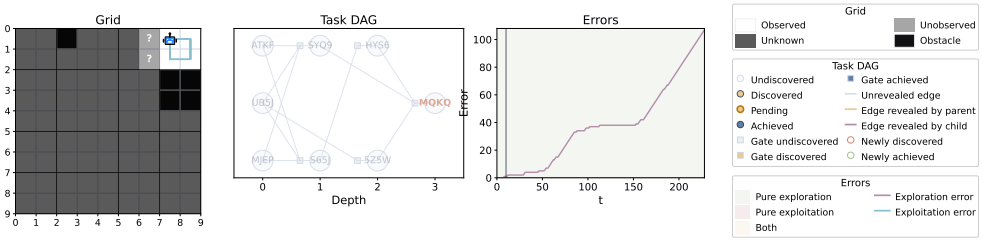


(e) GPT-OSS-120B, $t = 4$

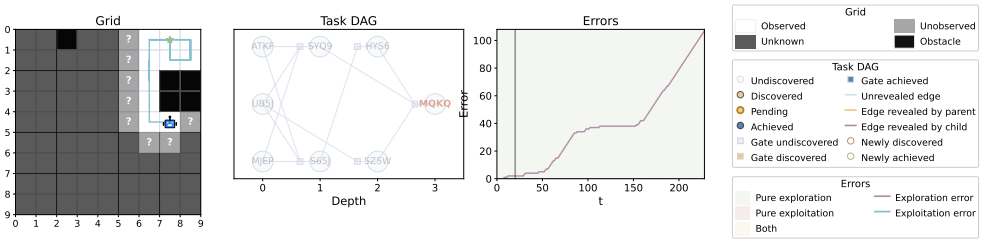
Figure 40: Results of action trajectories and metric values over each timestep for GPT-OSS-120B.



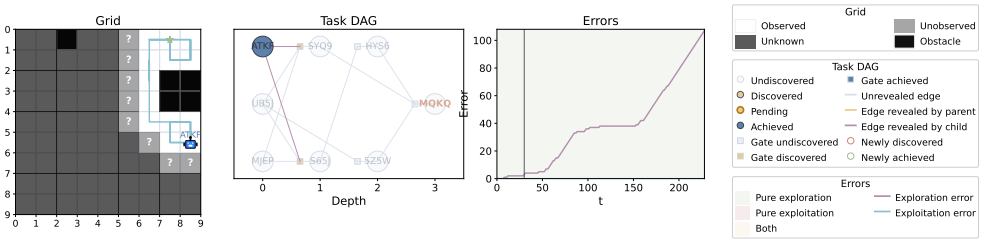
(a) GPT-OSS-120B, $t = 5$



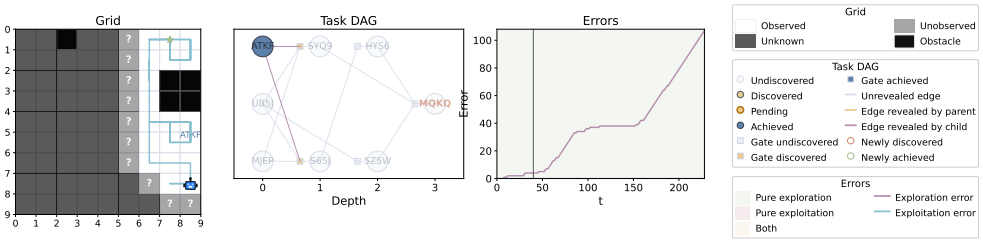
(b) GPT-OSS-120B, $t = 10$



(c) GPT-OSS-120B, $t = 20$

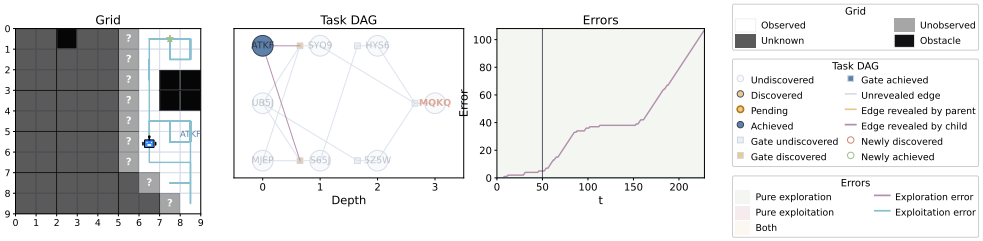


(d) GPT-OSS-120B, $t = 30$

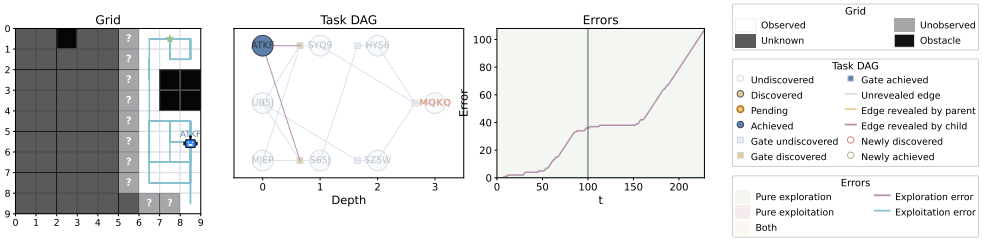


(e) GPT-OSS-120B, $t = 40$

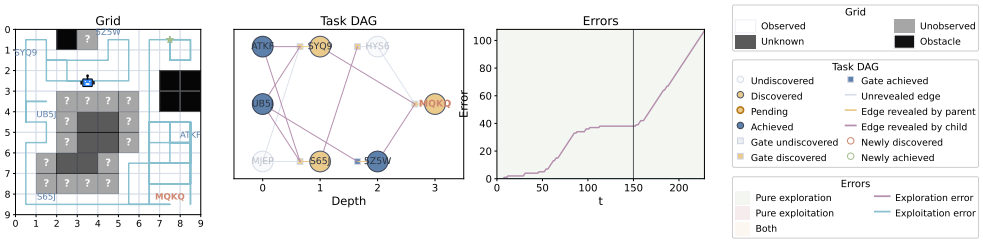
Figure 41: Results of action trajectories and metric values over each timestep for GPT-OSS-120B.



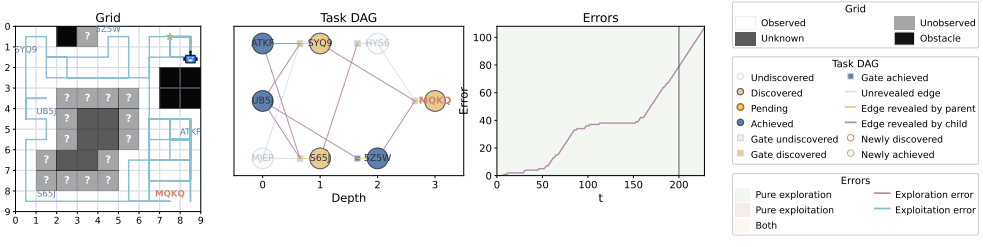
(a) GPT-OSS-120B, $t = 50$



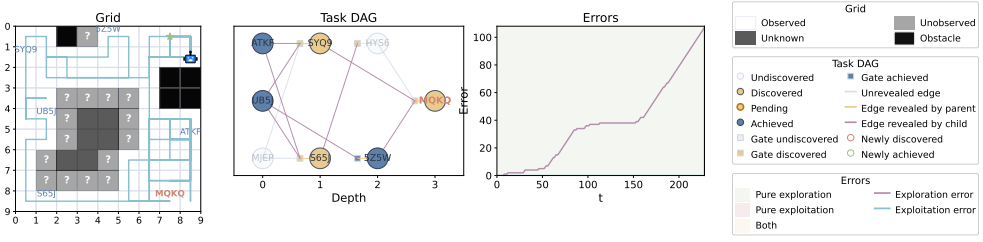
(b) GPT-OSS-120B, $t = 100$



(c) GPT-OSS-120B, $t = 150$



(d) GPT-OSS-120B, $t = 200$



(e) GPT-OSS-120B, $t = 228$

Figure 42: Results of action trajectories and metric values over each timestep for GPT-OSS-120B.

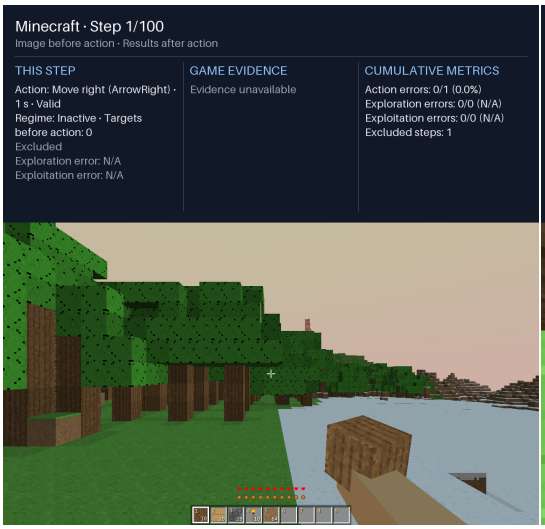
I EXTENSION TO VIDEO GAMES



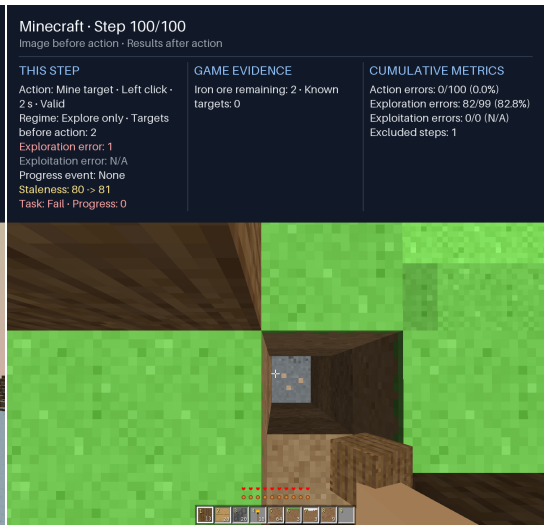
(a) Seed 0, Gemini 3.1 Flash Lite, Step 1/32



(b) Seed 0, Gemini 3.1 Flash Lite, Step 32/32



(c) Seed 0, GPT-4.1, Step 1/100



(d) Seed 0, GPT-4.1, Step 100/100

Figure 43: Observation examples of Minecraft Clone.

We used Minecraft Clone as a visually grounded environment (Ouyang et al., 2026), with agents selecting one action per decision. Minecraft requires navigating a three-dimensional block world to discover and collect resources. We discretized its keyboard and mouse controls into 21 predefined actions: movement in four directions, camera adjustment in four directions, jumping, mining, block placement, selection of one of nine inventory slots, and waiting. Movement lasted 1 second, jumping lasted 0.4 seconds, mining held the left mouse button for 2 seconds, and camera adjustments applied a fixed mouse displacement with a 0.2-second action budget. The simulation remained frozen between actions, preventing model inference time from affecting execution.

We evaluated Gemini 3.1 Flash Lite and GPT-4.1 under two visual memory conditions. With visual memory enabled, agents received the current screenshot and up to two previous screenshots; with visual memory disabled, they received only the current screenshot. Both conditions retained the prompts, actions, and rationales from the previous two decisions, isolating the contribution of historical screenshots. We ran each condition with nine environment seeds 0 to 8. The Minecraft task

required collecting at least two iron ore within 100 decisions; runs terminated early upon success. We report success rates across the nine seeds and compute exploration and exploitation error rates by pooling their respective error counts and scored opportunities across runs.