Exploration and Exploitation Errors Are Measurable for Language Model Agents

Jaden Park [*1] **Jungtaek Kim** [*1] **Jongwon Jeong** [*1] **Robert D. Nowak** [1 2] **Kangwook Lee** [2 3] **Yong Jae Lee** [1]

Abstract

Language Model (LM) agents are increasingly used in complex open-ended decision-making tasks, from AI coding to physical AI. A core requirement in these settings is the ability to both explore the problem space and exploit acquired knowledge effectively. However, systematically distinguishing and quantifying exploration and exploitation from observed actions without access to the agent's internal policy remains challenging. To address this intriguing issue, we design controllable environments inspired by practical embodied AI scenarios. Each environment consists of a partially observable 2D grid map and an unknown task directed acyclic graph. To enable policy-agnostic evaluation, we design a metric to quantify exploration and exploitation errors from agent's actions. We evaluate a variety of frontier LM agents and find that even state-of-the-art models struggle on our task, with different models exhibiting distinct failure modes.

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; 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 typically defined with respect to the agent's internal policy or value function. For LM agents, however, we typically have access only to the agent's observed actions. As a result, distinguishing and quantifying exploration and exploitation from behavior alone, without assuming a fixed strategy or access to the agent's internal policy, remains an open problem.

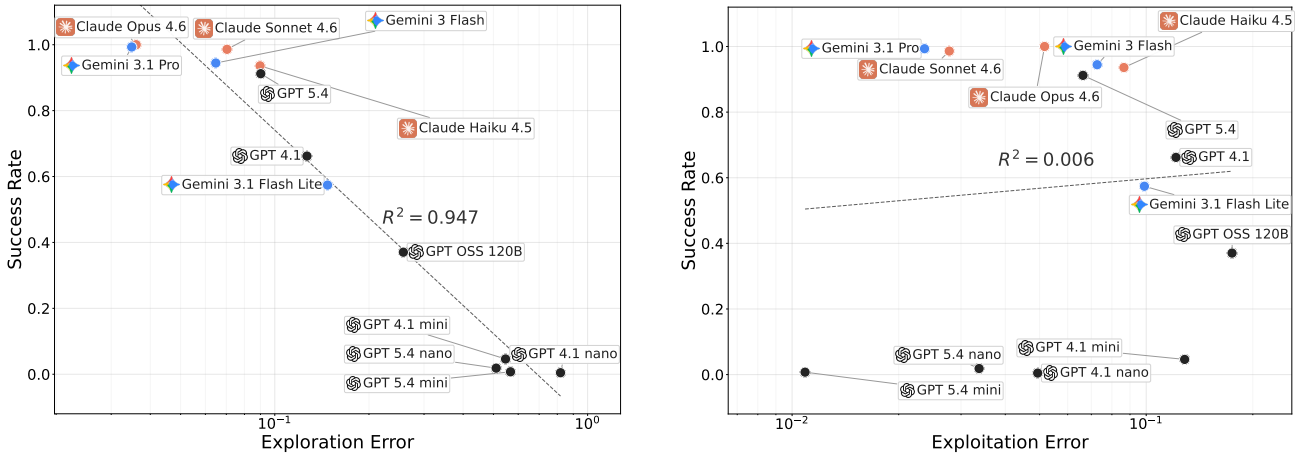
To address this, we introduce a policy-agnostic framework for quantifying exploration and exploitation errors from action trajectories alone. Our framework instantiates tasks as partially observable 2D grid maps paired with unknown task Directed Acyclic Graphs (DAGs) to enable systematic evaluation of exploration and exploitation from observed actions alone. This formulation captures the structure common to AI coding, workflow automation, and embodied AI: navigating a partially observed space while achieving tasks with complex dependencies. A key design choice in our environment is that we replace all semantic information in the task DAGs with symbolic representations. The agent must reason purely from the observed environment, effectively preventing conflation of semantic information from pretrained knowledge and semantic priors.

Within this framework, we define an error metric that 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 design is inspired by classical graph theoretic notions of redundancy and reuse (Whitney, 1932; Tarjan, 1972; Deng & Papadimitriou, 1999; Panaite & Pelc, 1999), and attributes each error to exploration, exploitation or both, depending on the map state.

With the proposed error metric, we evaluate a variety of frontier LM agents across diverse map configurations. We find that even state-of-the-art models often struggle on our task with different models exhibiting distinct failure modes.

[*]Equal contribution [1]University of Wisconsin–Madison [2]Ludo Robotics [3]KRAFTON. Correspondence to: Yong Jae Lee <yongjaelee@cs.wisc.edu>.

Presented at *the 43^{rd} International Conference on Machine Learning Workshop on Failure Modes in Agentic AI: Reproducible Triggers, Trace Diagnostics, and Verified Fixes*, Seoul, South Korea.



(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 Figure 1a, log exploration error and success rate show a strong negative linear relationship ($R^2 = 0.947$) whereas Figure 1b depicts a weak relationship ($R^2 = 0.006$) between log exploitation error and success rate.

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. More recently, Zhang et al. (2026) assessed whether LM agents can construct and exploit spatial beliefs through exploration.

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 finer-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. On the other hand, the behavior of LLM reasoning has been discussed in the work by Zeng et al. (2026), devising metrics based on the tree representation of reasoning traces.

Exploration and Exploitation. 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).

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 A.

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.

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

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

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

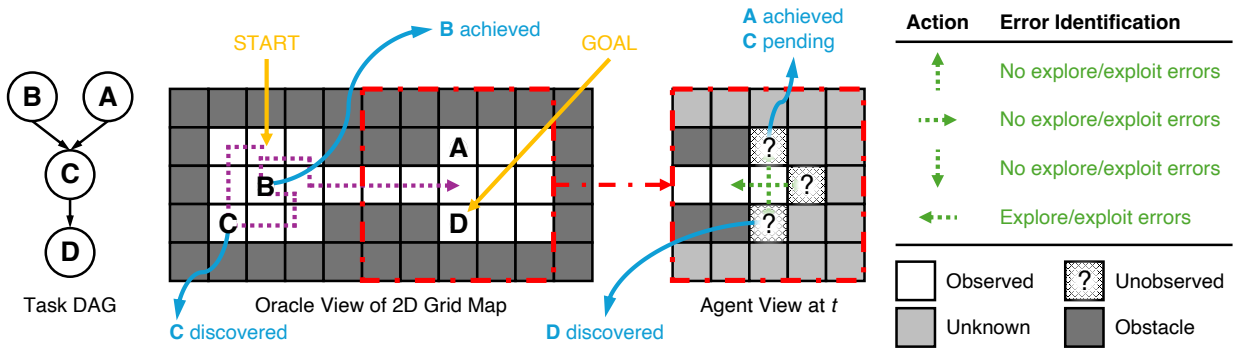


Figure 2. Illustration of the LM agent traversing the 2D grid map to achieve the goal given by the task DAG. In each timestep, the environment returns the set of admissible moves and the information about the task node, if discovered. The LM agent must retain relevant information about the map geometry,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

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 (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. 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 (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 (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.

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

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.

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 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 C, 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 (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

Table 1. Four cases for 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

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; refer to 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** (6) if it steps into a target cell or reduces the minimum distance to at least one of the target cells ($\text{Gain}(t\to t+1)=1$), and define the action as an error otherwise ($\text{Gain}(t\to t+1)=0$). However, 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 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 A.

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, define the error metric at timestep t:

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

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

5. Experimental Setup

We cover detailed settings for our experiments. For full details, refer to Appendices B.1 and B.2.

Task Generation. We generate task DAGs outline the 2D grid maps first. Then, we place the nodes on the 2D grid, by varying the following: (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 13 LMs spanning four model families: (i) **OpenAI ChatGPT**: GPT-4.1, GPT-4.1 mini, GPT-4.1 nano, GPT-5.4, GPT-5.4 mini, GPT-5.4 nano, (ii) **Google Gemini**: Gemini 3.1 Pro, Gemini 3 Flash, Gemini 3.1 Flash Lite, and (iii) **Anthropic Claude**: Claude Opus 4.6, Claude Sonnet 4.6, Claude Haiku 4.5. We also include (iv) **GPT-OSS-120B** to provide an open-weight baseline. For all models, we set the temperature as 0.

Prompts. We follow ReAct (Yao et al., 2023) framework. This prompt shares an identical environment description and action format. The prompt provides no strategic guidance, leaving the exploration–exploitation tradeoff entirely to the model's internal context and reasoning capabilities. Full details are provided in Appendix B.3.

Evaluation. We report **success rate**, **exploration error**, and **exploitation error**. Success rate measures the fraction

[4] For its detailed description, refer to Appendix A.

of episodes the agent achieves the goal within the 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.

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 a strong negative relationship between success rate and exploration error and a weak relationship between success rate and exploitation error. This aligns with the structure of our task because the LM agent must explore sufficiently to discover relevant task nodes before it can achieve the goal. As a result, persistent exploration failures will directly limit task completion. In contrast, low exploitation error does not correlate with success because an LM agent may still incur low exploitation error without exploring enough to uncover the nodes required for progress. Therefore, we argue the following finding:

> ***Finding 1***: Low exploration error rates are a strong predictor of success.

An interesting result is shown in Figure 3. While Claude Opus 4.6 and Gemini 3.1 Pro both achieve 100% success rate (Figure 1), the two models demonstrate different qualitative behaviors. As shown in Figure 3a, the fractions of exploration-only steps are similar in the early stage of the trajectories, but the two models diverge after the episode progresses to around 50%. Figure 3b further illustrates this difference qualitatively: Claude Opus 4.6 tends to head directly toward a goal node by exploiting known information, whereas Gemini 3.1 Pro continues exploring unobserved cells during its traversal toward the goal.

> ***Finding 2***: LM agents with similar success rates can exhibit qualitatively different behaviors.

As 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 C 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 significantly improves success rates, exploration and exploitation errors, and the average number of steps taken in successful trajectories (Table 2).

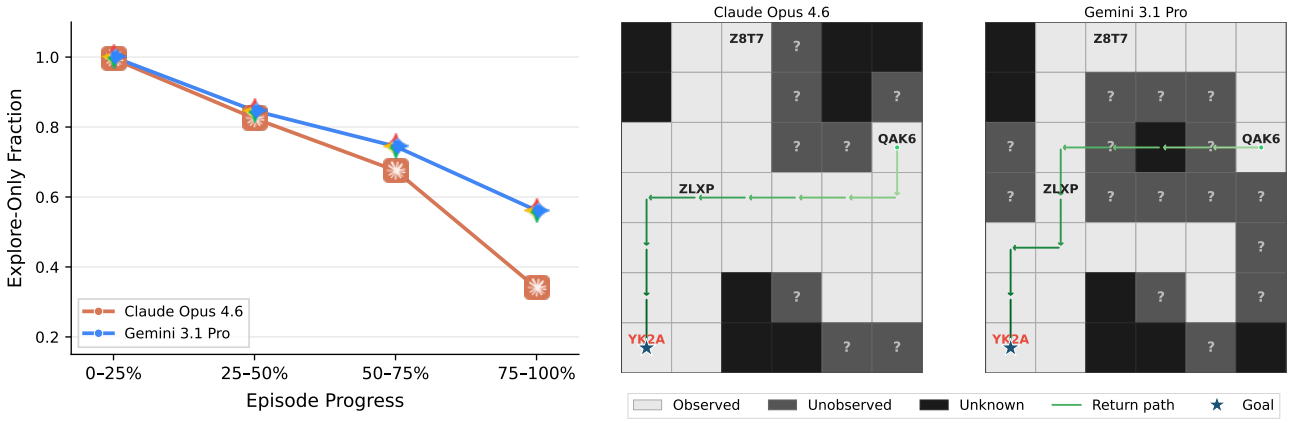
> ***Finding 3***: Harness engineering enhances the performance of LM agents utilizing historical outcomes.

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. 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. However, in some sense, our environment 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. Thus, our task is best viewed as a controllable test suite to measure raw exploration and exploitation capabilities that complements 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, 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.

Per-run variance and map difficulty. In addition to the variance in trajectories across different LM agents, we note that even for the same LM agent, different runs may induce substantially different trajectories, leading to different local scenarios and therefore different aggregated error values. While we run each experiment with three random seeds at



(a) Exploration Fraction vs. Progress

(b) Qualitative Example

Figure 3. Different exploration behavior between Claude Opus 4.6 and Gemini 3.1 Pro, both of which show success rate of 100%. In Figure 3a, until 50% of episode progress, two models show similar behaviors. Afterwards, however, Gemini 3.1 Pro performs more exploration than Claude Opus 4.6. Figure 3b illustrates this observation where Claude Opus 4.6 avoids stepping into unobserved cells despite the two paths incurring the same cost.

Table 2. Effect of harness engineering on success rates, exploration and exploitation errors, and the number of steps (of successful trajectories). All metrics are significantly improved for both Gemini 3.1 Flash Lite and GPT-4.1.

Model	Method	Success (%) ↑	Exploration Error ↓	Exploitation Error ↓	Steps ↓
Gemini 3.1 Flash Lite	Baseline	51.9	0.172	0.135	94.3
	+ Harness Engineering	**88.9**	**0.030**	**0.071**	**68.0**
GPT-4.1	Baseline	63.0	0.297	0.160	92.5
	+ Harness Engineering	**92.6**	**0.053**	**0.044**	**66.1**

temperature 0 to mitigate per-model and per-run variance, we expect cleaner trends with a larger experiment budget, especially since errors and success rates depend on multiple interleaved factors.

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 task DAGs, we evaluate a suite of frontier LM agents and show that low exploration error is strongly associated with success, while LM agents with similar success rates still exhibit qualitatively different behaviors. Our experiments show that harness engineering significantly affects how LM agents balance exploration and exploitation. Overall, our results provide a foundation for quantifying exploration and exploitation in LM agents beyond success rate, offering more informative lens for evaluating and improving LM agents in complex open-ended tasks.

Acknowledgements

This work was supported in part by National Science Foundation (NSF) award IIS-2404180, and Institute of Information & Communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) (No. 2022-0-00871, Development of AI Autonomy and Knowledge Enhancement for AI Agent Collaboration), and (No. RS-2025-2543949. Environment-Aware and Domain-Adaptive Multimodal Embodied AI for Real-World Interaction). This work was also supported by NSF award DMS-2023239, NSF CAREER award CCF-2339978, the Google Cloud Research Credits Program, and a grant from FuriosaAI. In addition, it used resources through CIS251382 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, supported by NSF awards OAC-2138259, OAC-2138286, OAC-2138307, OAC-2137603, and OAC-2138296.

Impact Statement

Although this paper focuses on LM agents in synthetic tasks, we acknowledge that ideas from this work can be used in real systems. Our environments and errors can help users find weak behavior before deployment, but they can also make unsafe agents look better if used alone. For real-world use, we recommend human review, task-specific safety checks, and hard action limits before agents run on their own.

References

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.

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

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Chen, Z., Du, W., Zhang, W., Liu, K., Liu, J., Zheng, M., Zhuo, J., Zhang, S., Lin, D., Chen, K., 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.

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

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

Ding, W., Tomlin, N., and Durrett, G. Calibrate-Then-Act: Cost-aware exploration in LLM agents. *arXiv preprint arXiv:2602.16699*, 2026.

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

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

Fraigniaud, P., Ilcinkas, D., and Pelc, A. Tree exploration with advice. *Information and Computation*, 206(11): 1276–1287, 2008.

Gubernatorov, K., Voronov, A., Voronov, R., Pasynkov, S., Perminov, S., Guo, Z., and Tsetserukou, D. Anywherevla: Language-conditioned exploration and mobile manipulation. *arXiv preprint arXiv:2509.21006*, 2025.

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

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

Inoue, Y., Misaki, K., Imajuku, Y., Kuroki, S., Nakamura, T., and Akiba, T. Wider or deeper? scaling LLM inference-time compute with adaptive branching tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Jeong, J., Kim, J., and Lee, K. 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.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. SWE-bench: Can language models resolve real-world Github issues? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Kim, J., Zeng, T., Lin, Z., Lee, M., Lee, C., Sohn, J.-y., Koo, H. I., and Lee, K. Transformers in the Dark: Navigating unknown search spaces via bandit feedback. *Transactions on Machine Learning Research*, 2026.

Krishnamurthy, A., Harris, K., Foster, D. J., Zhang, C., and Slivkins, A. Can large language models explore in-context? *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Lee, Y., Nair, R., Zhang, Q., Lee, K., Khattab, O., and Finn, C. Meta-Harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

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

Ma, C., Zhang, J., Zhu, Z., Yang, C., Yang, Y., Jin, Y., Lan, Z., Kong, L., and He, J. Agentboard: An analytical evaluation board of multi-turn LLM agents. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Merrill, M. A., Shaw, A. G., Carlini, N., Li, B., Raj, H., Bercovich, I., Shi, L., Shin, J. Y., Walshe, T., Buchanan, E. K., et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command kine interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

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

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

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

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Shridhar, M., Yuan, X., Cote, M.-A., Bisk, Y., Trischler, A., and Hausknecht, M. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

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

Szot, A., Kirchhof, M., Attia, O., and Toshev, A. Expanding LLM agent boundaries with strategy-guided exploration. *arXiv preprint arXiv:2603.02045*, 2026.

Tang, H., Hu, K., Zhou, J. P., Zhong, S., Zheng, W.-L., Si, X., and Ellis, K. Code repair with LLMs gives an exploration-exploitation tradeoff. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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

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

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

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

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Zeng, Y., Zhang, S., Kang, W., Wu, S., Zou, L., Fan, Y., Kim, H., Lin, Z., Kim, J., Koo, H. I., Papailiopoulos, D., and Lee, K. ReJump: A tree-jump representation for analyzing and improving LLM reasoning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026.

Zhang, P., Huang, Z., Wang, Y., Zhang, J., Xue, L., Wang, Z., Wang, Q., Chandrasegaran, K., Zhang, R., Choi, Y., 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. 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 **seen**.[6] Each node $u \in N$ has $\text{status}(u,t) \in \{\text{undiscovered}, \text{discovered}, \text{achieved}\}$ and $\text{type}(u) = \{\texttt{AND}, \texttt{OR}\}$ associated with it, which we define below:

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

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

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

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 $\text{type}(u)$.

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 $\text{progress}(g,t) = \text{achieved}$.

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

[6] Note that visited implies seen.

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

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.

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

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 3. 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.

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.

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.

Table 4. 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

Table 5. 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 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 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.

B. Additional Experimental Details

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

B.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 nodes of depth 0, we assume, without loss of generality, that task DAG contains exactly 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 apriori.

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

Table 6. 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 7. 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

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 for low, medium, and high exploitation, respectively. For corridor widths, we use 1, 1-3, 2-3 cells for low, medium, and high exploration. The cell in which 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 $\beta = 1$ for all settings. Finally, we fix `parent_depth_bias`$= 1$ for all tasks and use `option_count_probs` of $\{1:1.0\}$, $\{1:0.8, 2:0.2\}$, and $\{1:0.6, 2:0.4\}$ for easy, medium, and hard maps while both `dependency_count_probs` and `goal_dependency_count_probs` are uniformly sampled from $\{1,2\}$, $\{1,2\}$, and $\{1,2,3\}$, respectively.

B.2. Statistics on Experiments

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

Table 8. 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

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.

B.3. Prompts

Each system prompt consists of three parts: (1) an environment description, (2) a strategy prompt, and (3) an action format specification. We also provide example turns illustrating the observation format.

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

```
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"}
```

C. Harness Engineering

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.

D. Declaration of LLM Usage

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.