# GraphMind: LLMs as Dynamic Knowledge Builders for Sequential Decision-Making

**Sunguk Shin**[1*], **Hayeong Lee**[1*], **Jun Ho Seo**[1*], **Jinho Lee**[1],
**Myunsoo Kim**[1], **Minsuk Chang**[2], **Byung-Jun Lee**[1,3†]
[1]Korea University [2]Google DeepMind [3]Gauss Labs Inc.
{ssw1419,hayeong_lee,junhoseo,jinho0997,m970326,byungjunlee}@korea.ac.kr
minsukchang@google.com
[*]Equal contribution

## Abstract

While the reasoning capabilities of large language models (LLMs) have advanced considerably, efficiently internalizing and leveraging new information in dynamically interactive environments remains a significant challenge. This limitation is particularly pronounced in partially observable environments, which require agents to manage long-term memory and perform effective exploration under incomplete information. To address this, we propose an LLM agent architecture that integrates a knowledge graph as a graph-based memory module. The agent incrementally constructs the knowledge graph through environmental interactions and retrieves relevant information to generate efficient plans. We evaluate our approach in complex navigation tasks specifically designed to present long-horizon and partially observable challenges. Experimental results demonstrate that incorporating a self-extending memory module significantly enhances the performance and efficiency of the LLM's planning capabilities.

## 1 Introduction

Large language models (LLMs) have demonstrated remarkable performance in natural language understanding and generation, establishing themselves as foundational tools across a wide range of domains. Recently, research has increasingly focused on leveraging LLMs for interaction with dynamic environments, exploiting their strong prior knowledge and reasoning capabilities. Studies such as Carta et al. (2023); Paglieri et al. (2024); Lee et al. (2025) have reported promising results in sequential decision-making tasks, highlighting the potential of LLMs as agents in complex, interactive settings. These approaches typically rely on a short context window over recent trajectories, limiting the agent's ability to retain and exploit long-term context during decision-making.

However, many real-world tasks are inherently long-horizon and complex; the added challenge of partial observability necessitates that agents maintain and reason over extended context to facilitate effective decision-making. Figure 1 illustrates an environment that presents substantial difficulties for a naive LLM planner, especially under such observational constraints. While Retrieval-Augmented Generation (RAG) frameworks (Lewis et al., 2020; Yu et al., 2022; Han et al., 2024; Shim et al., 2026) successfully leverage large-scale external corpora, these methods are often limited by the static nature of their knowledge sources. Specifically, fixed documentation cannot account for the non-stationary, incrementally expanding information generated through continuous environmental interaction. This limitation motivates the development of new mechanisms for adaptive memory construction and retrieval tailored to sequential decision-making under partial observability.

Another line of work explores expanding external memory to support long-horizon tasks. For instance, studies by Anthropic (2025) and Comanici et al. (2025) utilize the challenging benchmark of Pokémon Red to evaluate long-term memory in LLMs. Instead of persistently including all information in the prompt, both approaches equip the agent with tools for on-demand knowledge retrieval, enabling a form of extended reasoning that streamlines the decision-making process. Specifically, the approach by Anthropic (2025) with Claude Opus 4 maintains external memory files to store key information. Similarly, the approach by Comanici et al. (2025) with Gemini-2.5-Pro condenses action sequences in batches to reduce input tokens. This summarization focuses on event sequences rather than constructing a spatial mental map. While maintaining continuity, both approaches result in an inefficient memory structure and substantial storage requirements, limiting scalability.



*Findings of the Association for Computational Linguistics: ACL 2026*, pages 32985–33007

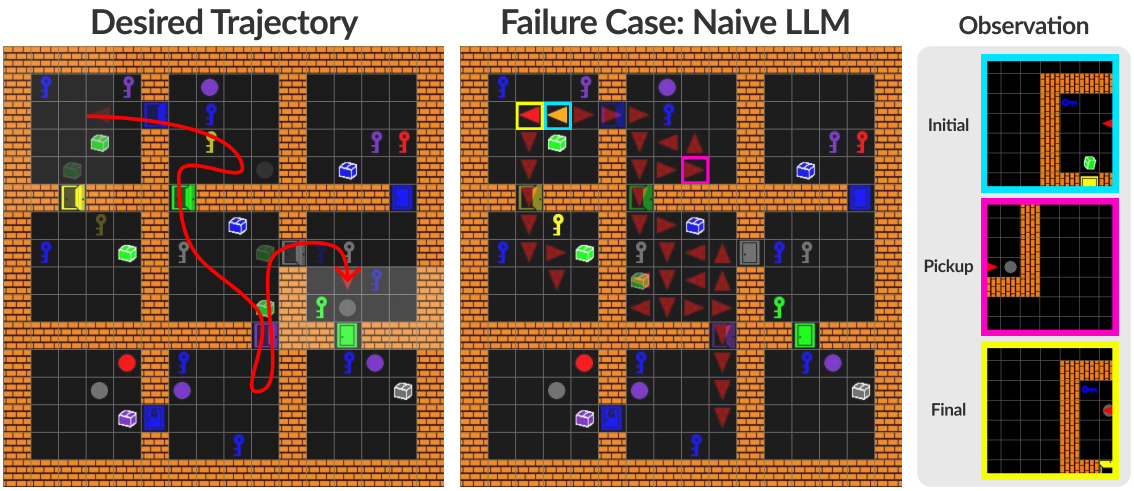



Figure 1: Examples of a desired trajectory (**left**) and a failed trajectory (**middle**) in the partially observable mission "put a gray ball next to the green key." The agent moves from the start position (▸) to the final position (▸). The naive LLM planner fails to complete the mission due to insufficient exploration.

To mitigate these issues, we propose *GraphMind* [1], a scalable and effective self-expanding external memory and planning framework with two key components. First, we organize information from past interactions into a graph-based representation which provides a compact yet expressive memory mechanism, particularly advantageous in partially observable navigation tasks, as it explicitly captures spatial relationships between objects and locations (Han et al., 2024). Second, to support efficient decision-making, we augment the planning capabilities of LLMs with a domain-specific language to enable structured reasoning and planning to improve exploration efficiency, while grounding the knowledge graph in trajectories collected through the actor's behavior. Our experiments show that the proposed structured approach enables an LLM agent to complete tasks in challenging long-horizon, partially observable environments.

## 2 Domain and Problem Statement

Complex, long-horizon tasks are common in real-world settings and require both effective memory mechanisms and advanced planning capabilities (Kim et al., 2025; Hu et al., 2025; He et al., 2025). Especially, we focus on enhancing LLM's planning capabilities in partially observable environments that provide only limited local information. Consequently, in contrast to alternatives such as McDermott (2000); Shridhar et al. (2020); Samvelyan et al. (2021), we adopt and extend BabyAI (Chevalier-Boisvert et al., 2019), a partially observable 2D gridworld characterized by diverse challenges where agent interactions drive dynamic environmental transitions. These challenges include object manipulation, navigation, exploration–exploitation trade-offs, and mission execution guided by instructions in natural language. In addition, BabyAI provides an Oracle Solver (referred to as BOT in (Chevalier-Boisvert et al., 2019)), which generates step-by-step solutions using hand-coded rules and an internal stack machine. This feature enables systematic evaluation and analysis of how our method expands memory and influences planning in long-horizon, partially observable environments.

Each environment layout consists of $n$ rooms connected by colored doors and populated with color-coded objects. The agent explores these rooms to locate target objects, while its field of view is restricted by walls and doors. Locked doors require keys of the corresponding color. To further emphasize partial observability, we modified the environment such that open doors also block the line of sight, preventing perception beyond the doorway.

In all experiments, we evaluate our approach using four-room and nine-room layouts, each comprising 25 cells (excluding walls). The nine-room ($3 \times 3$) configuration corresponds to the BossLevel, which represents the most challenging task in BabyAI. In our experimental setup, we focus on two challenging missions, `OpenDoor` and `PutNextTo`, that have proven difficult in prior work (Carta et al., 2023). The detailed discussion of the

[1]Our code is available at: https://github.com/ku-dmlab/GraphMind.

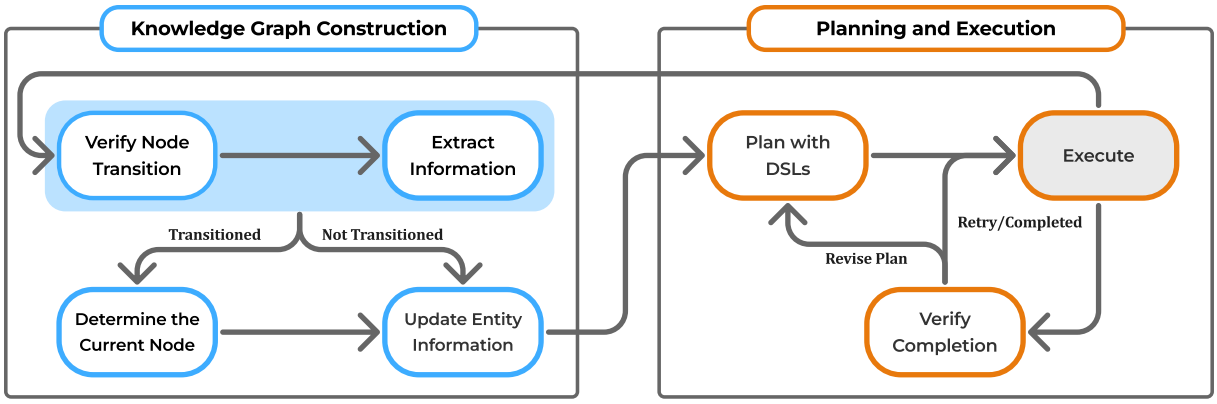



Figure 2: Overview of the proposed framework. The left panel illustrates the construction of the knowledge graph, where the agent verifies node transitions, extracts information from observations, determines the current node, and updates entity information. The right panel depicts the planning and execution loop, in which the LLM generates DSL-based plans, the actor executes instructions, completion is verified, and plans are revised if necessary. Arrows indicate the flow of information and iterative feedback between modules.

multimodal observations and environment is provided in Appendix D and Appendix B, respectively.

## 3 Proposed Approach

We propose *GraphMind*, a framework that enables agents to operate effectively in partially observable environments. GraphMind dynamically constructs a self-expanding graph-based memory from collected observations and employs adaptive planning to retrieve and exploit the knowledge required for task completion. Graph structures provide a compact and expressive way to encode large-scale, heterogeneous, and relational information, in contrast to linear structures. These structures are particularly well-suited for capturing spatial relationships, which are critical in navigation tasks (Hogan et al., 2021; Han et al., 2024).

The proposed framework operates in two iterative stages: (1) **Knowledge Graph Construction**, where observations are integrated into an incrementally expanding graph representation (Section 3.1), and (2) **Planning and Execution**, where a domain-specific language (DSL) supports structured reasoning and action selection to gather task-relevant information under partial observability (Section 3.2). This iterative process enables systematic expansion of the agent's knowledge base and progressively improves task performance. An overview of our approach is shown in Figure 2, highlighting the interaction between graph construction and planning.

### 3.1 Knowledge Graph-based Memory

We propose a graph-based memory representation that expands dynamically with the help of LLMs. The graph expands incrementally as the agent explores, with LLM-based modules guiding construction. Nodes correspond to objects or locations, and edges encode relationships through relational predicates. This design allows the agent to capture both spatial and semantic information, which are essential for reasoning and planning in partially observable environments. We focus on four relation types: is_connected_with (spatial connectivity), contains (object presence within a space), is_located_to (directional attributes), and are_apart (relative distances that help distinguish otherwise identical objects). Figure 3 shows an example of constructing such a graph from agent observations.

Figure 2 (left) illustrates knowledge graph construction, implemented via four dedicated modules:

- **Verify Node Transition**: Analyze the agent's action sequence to detect transitions and update is_connected_with relations.
- **Extract Information**: Parse the current observation to identify objects, expand contains relations, and annotate objects with directional and distance attributes to establish is_located_to and are_apart.
- **Determine Current Node**: Localize the agent by integrating observed entities, verified transitions, and connections to the previously known location.
- **Update Entity Information**: Merge extracted attributes into the current node to maintain a coherent, up-to-date representation of the environment.

This process yields an adaptive memory structure that evolves continuously during interaction,

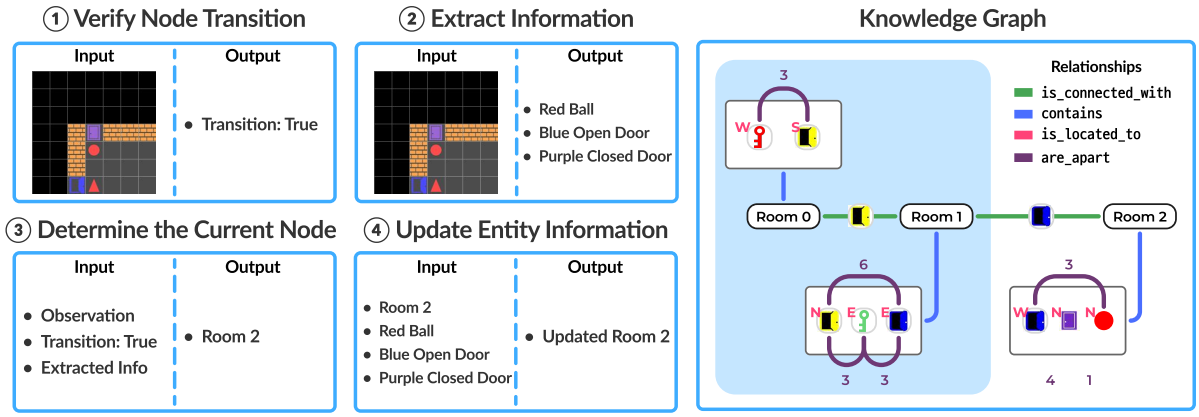



Figure 3: Example of updating the knowledge graph in BabyAI. When the agent (▶) observes a red ball (●) and a purple door (■) after traversing a blue door (■), the corresponding modules verify the door traversal, extract information, and expand the graph, and the associated relationships, building upon the existing graph (shaded area).

supporting long-horizon reasoning. Detailed descriptions of each module and the corresponding prompts are provided in the Appendix A.

**External Tools for Knowledge Utilization** We augment the LLM with external tools to retrieve relevant information from a knowledge graph, enabling effective utilization of structured environmental knowledge. The LLM determines whether to retrieve information based on its current observation, avoiding the inclusion of the full summarized memory in every prompt, following prior approaches (Anthropic, 2025; Comanici et al., 2025). The tools return information regarding the specified and adjacent rooms, as well as the shortest path from a given location to the nearest target via a search algorithm, such as Breadth-First Search. This allows the agent to efficiently manage a scalable memory that expands dynamically as the agent explores the environment. A detailed description of the designed tools is provided in Appendix A.3.

### 3.2 Planning using Domain-specific Language

A challenge in LLM-based agents lies in bridging the gap between high-level planning and low-level action execution. LLMs excel at generating symbolic and abstract plans but often lack the precision and reliability required for fine-grained control in dynamic environments (Ma et al., 2024; Wen et al., 2024). In contrast, low-level action policies—whether heuristic controllers or reinforcement learning agents—are effective at executing primitive behaviors but lack the ability to reason about long-term dependencies or abstract objectives. Our framework addresses this disconnect through the use of domain-specific languages (DSLs), which provide a structured interface between the symbolic reasoning of the LLM and the concrete action space of the environment. Due to their ability to bridge these domains, DSLs have been widely adopted to enhance the problem-solving capabilities of LLMs (Barke et al., 2024; Chollet et al., 2024).

Table 1: DSLs and their descriptions for BabyAI.

| DSLs | Description |
|---|---|
| `find_door()` | Find a door in the current room. |
| `pass_door(x)` | Pass through door x. |
| `go_to_entity(x)` | Move to face entity x. |
| `pick_up_entity(x)` | Pick up entity x. |
| `drop_entity(x)` | Drop the currently held entity x on an empty cell. |
| `drop_next_to_entity(x,y)` | Drop the currently held entity x next to entity y. |

The DSL provides a compact yet expressive action space, enabling the agent to efficiently navigate, manipulate objects, and execute high-level strategies required for task completion. In our framework, by encoding navigation and interaction primitives as DSL instructions (see Table 1), allowing the LLM to function as a high-level controller. This hierarchy enables the model to focus on strategic goal-setting while delegating granular execution details to a low-level actor. This separation of concerns reduces the cognitive load on the LLM and enhances robustness in action execution. Consequently, our approach effectively aligns symbolic reasoning with embodied interaction, bridging a longstanding gap between high-level planning and low-level control in partially observable environments. The prompting templates are provided in Appendix A.

We formalize the decision process as an iterative "plan–execute–verify–revise" loop, as illustrated in Figure 2 (right). This cyclical structure ensures that

Table 2: Success rate of completed missions (`PutNextTo` and `OpenDoor`). For graph edit distance (GED), lower values indicate a more accurately constructed knowledge graph. We adopted expert heuristic bot as a downstream actor. We denote our framework variants as follows: with the knowledge graph (KG), with the stacked memory (SM), and without the memory (w/o Memory). We report mean success rate and their 1 standard errors (SE).

| Mission | # rooms | Metrics | Gemini-2.5-Flash KG | Gemini-2.5-Flash w/o Memory | Gemini-2.5-Flash SM | Gemini-2.5-Pro KG | Gemini-2.5-Pro w/o Memory | Gemini-2.5-Pro SM |
|---|---|---|---|---|---|---|---|---|
| Put Next To | 2×2 | Success (%) | **96.7** ± 3.3 | 90.0 ± 5.6 | 87.7 ± 6.3 | 83.3 ± 6.6 | 73.3 ± 8.2 | **90.0** ± 5.6 |
| | | GED | 4.97 ± 1.19 | — | — | 3.43 ± 1.06 | — | — |
| | 3×3 | Success (%) | **66.7** ± 8.8 | 36.7 ± 8.9 | 36.7 ± 8.9 | **70.0** ± 8.5 | 56.7 ± 9.2 | 43.3 ± 9.2 |
| | | GED | 8.43 ± 1.62 | — | — | 6.33 ± 1.45 | — | — |
| Open Door | 2×2 | Success (%) | **100.0** ± 0.0 | 86.7 ± 6.3 | **100.0** ± 0.0 | **100.0** ± 0.0 | 93.3 ± 4.6 | **100.0** ± 0.0 |
| | | GED | 1.13 ± 0.36 | — | — | 1.60 ± 0.34 | — | — |
| | 3×3 | Success (%) | **70.0** ± 8.5 | 66.7 ± 8.8 | **70.0** ± 8.5 | **83.3** ± 6.8 | 73.3 ± 8.2 | 76.7 ± 7.9 |
| | | GED | 9.70 ± 1.87 | — | — | 3.60 ± 0.82 | — | — |

planning remains adaptive, resilient to execution errors, and robust under partial observability.

- **Plan with DSLs**: The LLM generates a sequence of DSL instructions conditioned on the current observation and retrieved knowledge graph information accessed via external tool calls.
- **Execute**: The actor executes the next DSL instruction.
- **Verify Completion**: The system verifies whether the instruction has been achieved.
- **Revise Plan**: If verification fails, the framework either generates a revised set of DSL instructions or allows the actor to retry the current instruction.

## 4 Experiments

Our experiments were conducted in complex multi-room environments, in contrast to the simple, single-room setups used in prior work (Carta et al., 2023; Paglieri et al., 2024). To effectively evaluate exploration efficiency, we filter layouts to guarantee that accessing the target objects requires obtaining a key to unlock a door. In particular, for $3 \times 3$ layouts, we enforce that completing the mission necessitates exploring at least four rooms. We provide our layout generation procedure in Appendix B.1.

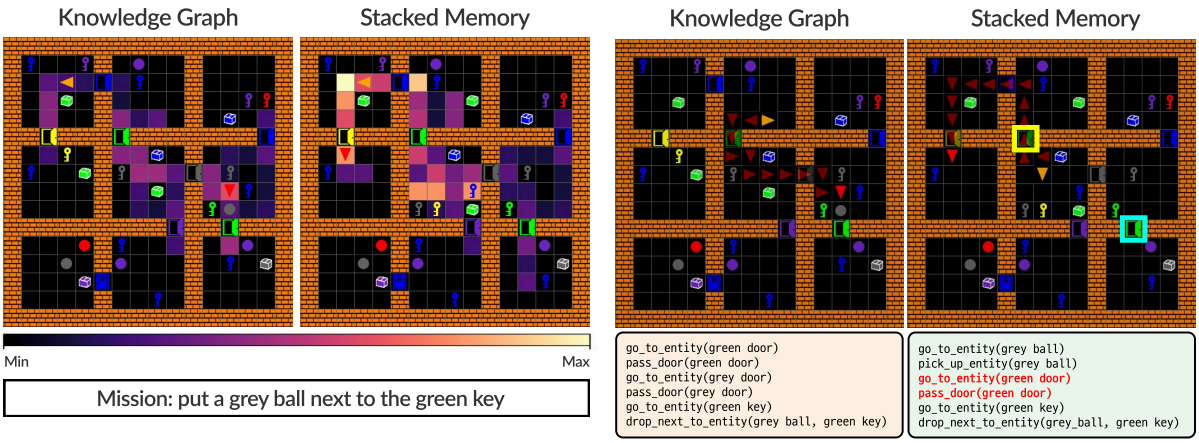
We evaluated two models from the Google DeepMind Gemini 2.5 series (Comanici et al., 2025): Gemini-2.5-Flash and Gemini-2.5-Pro. To account for the non-stationarity of partially observable environments and the variability of LLM responses, we conducted three trials for each layout. While our experiments evaluate the LLM's planning capabilities by requiring the agent to explicitly invoke external tools for memory retrieval, we also provide baseline results for a static memory setting where the agent has continuous access to the full graph information in Appendix F.

Table 3: Success rate of `PutNextTo` mission. While LLM-as-Agent struggles due to inevitable hallucinations, our method was able to solve some tasks.

| # rooms | Metrics | Gemini-2.5-Flash Bot | Gemini-2.5-Flash LLM | Gemini-2.5-Pro Bot | Gemini-2.5-Pro LLM |
|---|---|---|---|---|---|
| 2×2 | Success (%) | 96.7 ± 3.3 | 20.0 ± 7.4 | 83.3 ± 6.6 | 13.3 ± 6.3 |
| | GED | 4.50 ± 1.65 | 5.33 ± 1.10 | 3.17 ± 1.56 | 8.90 ± 1.13 |
| 3×3 | Success (%) | 66.7 ± 8.8 | 20.0 ± 7.4 | 70.0 ± 8.5 | 10.0 ± 5.6 |
| | GED | 7.80 ± 1.63 | 6.80 ± 1.06 | 6.00 ± 1.83 | 9.2 ± 0.95 |

To evaluate the efficacy of our knowledge graph as a memory module, we adopted two additional baselines. The first, **without Memory (w/o Memory)**, removes the knowledge graph construction stage entirely. In this setting, the agent still leverages the same DSL for planning and exploration, serving as a baseline to measure the direct contribution of our framework (**KG**). The second, **Stacked Memory (SM)**, replaces the graph-based memory with a linear, stack-structured alternative to assess the role of memory topology. In this baseline, at each step, the output of the **Extract Information** module is appended to a linear data store.

We analyze the effectiveness of LLMs in leveraging external modules for memory storage and retrieval in a partially observable environment using a planning DSL. The actor interacting with the environment is instantiated in two variants: (1) an expert heuristic actor, which isolates the contributions of memory and planning from the variability in action execution, and (2) an LLM-as agent, which introduces additional considerations due to the inherent uncertainty of LLM outputs.




(a) Cell visitations under Gemini-2.5 Flash planner.

(b) Plans generated by Gemini-2.5 Flash planner.

Figure 4: Comparison of a **knowledge graph (KG)** versus **stacked memory (SM)** for a Gemini-2.5-Flash planner. The KG's structured representation enables efficient exploration **(a)** and results in correct plan **(b)**. In contrast, the SM leads to confused exploration and planning failure, as the agent cannot distinguish between two identical doors.

### 4.1 Effectiveness of External Memory under Partially Observability

We evaluate our framework on two BabyAI missions with results presented in Table 2. The full framework (KG) consistently outperforms w/o Memory, highlighting the critical role of structured external memory in enabling effective agent behavior. This is particularly evident in the $3 \times 3$ setting, which requires a longer planning horizon and strategic exploration. While Gemini-2.5-Flash achieves a success rate of 63.3% (w/o Memory), KG increases performance to 81.7%. This substantial 18.4%p improvement demonstrates that the external memory enables more effective problem-solving by providing structured context. Qualitative examples of these execution are provided in Appendix E.

To evaluate the extendability of our framework to non-expert implementations of DSLs, we adopt an LLM-as-agent approach (Paglieri et al., 2024). Compared to the rule-based heuristic actor, the LLM-based implementation exhibits a significant performance degradation, as shown in Table 3. This outcome is expected, as LLMs are prone to hallucinations (Kalai et al., 2025) and struggle to construct a coherent internal model from egocentric observations (Yang et al., 2025). We view the automatic discovery of such functions as an important direction for future research.

**Scalability to Larger Environments** To further validate the scalability of our framework, we extend our evaluation to larger room layouts, specifically $4\times4$ (16 rooms) and $7\times7$ (49 rooms) configurations. We conducted 36 trials per room size with a maximum episode horizon of 400 steps to accommodate the increased environment complexity. Table 4 summarizes success rates and average steps to completion for KG and SM under these settings.

Table 4: Scalability evaluation on larger room layouts (PutNextTo mission, Gemini-2.5-Flash). We report success rate (%) and average steps to completion. KG consistently outperforms SM, with the performance gap widening as environment size increases.

| Method | Success (%) | | Avg. Steps | |
|---|---|---|---|---|
| | $4\times4$ | $7\times7$ | $4\times4$ | $7\times7$ |
| SM | $33.3 \pm 7.9$ | $33.3 \pm 7.9$ | 230.6 | 305.1 |
| KG (Ours) | $\mathbf{72.2 \pm 7.5}$ | $\mathbf{83.3 \pm 6.2}$ | **190.1** | **157.1** |

As shown in Table 4, KG significantly outperforms SM across both larger settings, achieving a 44.5%p gain in the $7\times7$ environment (33.3% $\rightarrow$ 83.3%). Notably, while SM exhibits near-constant failure rates regardless of scale, KG performance *improves* in the larger setting, suggesting that the structured memory becomes increasingly effective as the environment grows. This is further corroborated by the average step counts: KG solves tasks in substantially fewer steps than SM (157.1 vs. 305.1 in $7\times7$), demonstrating that persistent graph-based memory enables more efficient exploration and planning as the task complexity increases.

### 4.2 Effectiveness of Graph-based Memory Structure

We evaluate the effectiveness of a graph-based memory structure by comparing it against a stacked memory alternative to examine whether the structure of external memory influences performance.

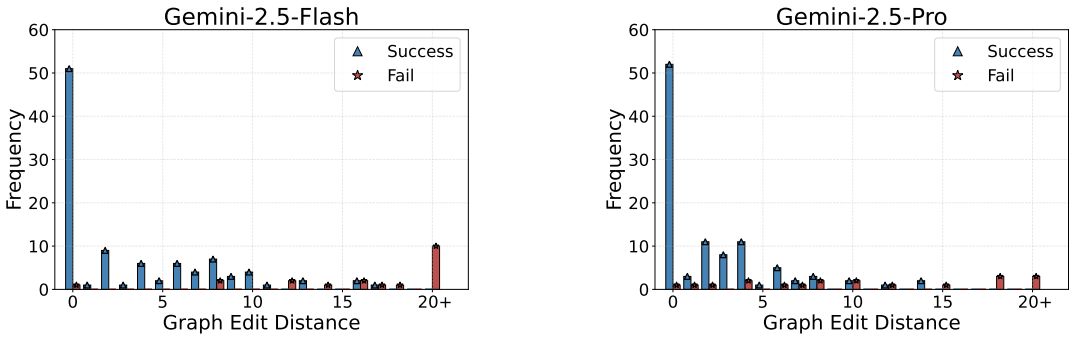



Figure 5: The distribution of GED scores for task successes (blue triangles) and failures (red stars), using Gemini-2.5-Flash and Gemini-2.5-Pro. GED scores exceeding 20 are aggregated into the 20+. Successful episodes are concentrated at lower GED values, whereas failures are distributed across a wider range of higher distances.

As illustrated in Table 2, the agent equipped with SM fails to navigate the environment efficiently. Specifically, the SM-based agent degrades significantly due to error accumulation when processing extended context windows. As illustrated in Figure 4a, the agent frequently revisits already explored cells and repeatedly rediscovers objects. In contrast, the knowledge graph-based agent exhibits more structured and efficient exploration patterns, despite the inevitable redundancies caused by partial observability. Moreover, leveraging the knowledge graph-based memory facilitates efficient pathfinding between mission-critical objects, such as “a grey ball" and “a green key".

The inefficiency is further underscored by the suboptimal plan sequences generated by the planner. Figure 4b shows examples of LLM-generated plans along with the corresponding execution trajectories for our method and the baseline. The SM-based agent confuses the green door in the lower-right (blue highlight), with the green door (yellow cells 2 and 5). This confusion suggests that SM is an ineffective strategy for managing a dynamically growing memory. In contrast, the agent equipped with the knowledge graph successfully planned a trajectory to reach the target object.

Simply using external memory does not guarantee improved performance; indeed, a naive approach can be **detrimental**. Our results show that a graph-based memory enhances agent efficiency, while an unstructured memory architecture can impair performance. Furthermore, we provide a comparative analysis of the token-level cost for various memory structures in Appendix C.

### 4.3 Evaluating LLM-Constructed KGs

A key challenge in constructing graph-based memories is the consistent tracking of entities over time, particularly in dynamic environments where object relocation introduces significant state variability. To evaluate whether LLM-constructed knowledge graphs provide meaningful support for planning, we adopt Graph Edit Distance (GED) (Sanfeliu and Fu, 2012) which quantifies structural similarity to the ground-truth by calculating the minimum number of edit operations required for alignment. Intuitively, a graph that more closely aligns with the true environment should facilitate more accurate reasoning and planning.

As shown in Table 2, the average GED increases with environment scale due to compounded inference errors. Figure 5 demonstrates a strong negative correlation between GED and task success in the `PutNextTo` mission. Successful trials maintain low averaged GED values of 3.62 ($2 \times 2$) and 3.25 ($3 \times 3$), whereas failed trials exhibit higher averages of 30.0 and 16.9, respectively. Notably, the success rate reached 98.10% when the knowledge graph was highly accurate, but plummeted to 0% as error accumulation pushed the GED beyond a threshold of 20.

These findings support the intuition that structural fidelity is critical: the closer the constructed graph aligns with the ground truth, the more effectively it guides sequential decision-making. Figure 6 demonstrates typical failure modes; significant changes in observations often caused the LLM to misclassify revisited rooms as novel, leading to degraded accuracy. This suggests that while LLMs possess an intrinsic capacity for knowledge utilization, performance remains sensitive to the structural accuracy of the underlying graph.

## 5 Discussion

**Computational Cost vs. Performance Trade-off** GraphMind incurs higher per-step inference

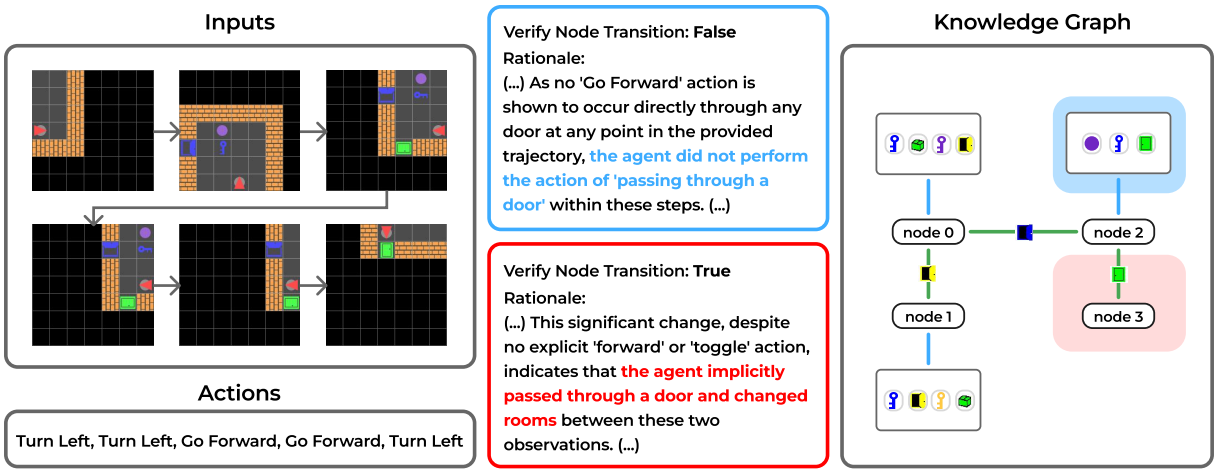



Figure 6: Examples of planner correctly (**Blue Box**) and incorrectly (**Red Box**) verifies whether the agent has transitioned into another node. LLM planner confuses observation changes due to rotation to node changes, leading to spurious node expansion (**node 3**).

cost than simpler baselines due to the overhead of knowledge graph construction and the iterative plan–execute loop. As shown in Appendix C, the total token consumption of our method is approximately 1.6× that of the stacked memory (SM) baseline. However, this overhead is offset by a substantial gain in task performance (18.4%p improvement) and a reduction in the number of steps required to complete tasks.

Notably, SM already employs chain-of-thought (CoT)-style reasoning, either via prompting or reasoning models; the key distinction is that SM accumulates extracted observations into a flat linear store, whereas GraphMind explicitly organizes them into a structured graph through the LLM-driven construction loop. Therefore, Appendix C can be interpreted as a token-level comparison between CoT-style reasoning without structured graph expansion (SM) and our approach. This suggests that the additional cost of maintaining a structured memory becomes increasingly justified—and indeed necessary—as task complexity and partial observability grow.

**LLM Hallucination and Action Misalignment** As shown in Table 3, replacing the heuristic actor with an LLM-as-agent leads to substantial performance degradation. The primary failure mode is *action misalignment*: the LLM-as-agent fails to faithfully execute the designated DSL instruction despite correct high-level planning. For example, given the instruction `drop_next_to_entity(purple_box, yellow_box)`, the agent may reduce the distance between the two objects without satisfying the adjacency constraint. This behavior stems from the difficulty LLMs face in constructing a coherent spatial model from egocentric observations (Yang et al., 2025), compounded by their susceptibility to hallucinations (Kalai et al., 2025). Furthermore, action misalignment induces redundant observations that are passed into KG construction, leading to higher GED and cascading errors downstream. In contrast, when a heuristic bot is employed, errors tend to remain localized: KG construction failures typically manifest as structural redundancies (e.g., duplicating an existing room node) rather than fundamental corruption of the graph topology, allowing subsequent correct planning to recover the intended trajectory. Mitigating action misalignment through parameter-efficient fine-tuning or finer-grained DSL primitives remains an important direction for future work.

## 6 Related Work

**LLMs as Agents in Sequential Decision-Making** Large language models (LLMs) have demonstrated strong performance across several challenging tasks, including question answering (Rajpurkar et al., 2016), mathematics (Hendrycks et al., 2021), and complex iterative interactions. For instance, Ma et al. (2024) achieved notable results in the real-time strategic decision-making environment.

Furthermore, Paglieri et al. (2024) benchmarked LLM-as-Agent approaches across several game-based environments, including BabyAI (Carta et al., 2023), TextWorld (Côté et al., 2018), Baba Is AI (Cloos et al., 2024), MiniHack (Samvelyan et al., 2021), and NetHack Learning Environment (NLE) (Küttler et al., 2020). However, LLM-as-agent ap-

proaches exhibit limitations in long-context scenarios, particularly in tasks that require effective utilization of historical information.

**Addressing Hallucination via Knowledge Retrieval** Although the ability of LLMs to handle long contexts has improved, they still suffer from hallucinations, which is a critical issue in long-context problems, such as sequential interactions with an environment. To mitigate this, prior work has proposed Retrieval-Augmented Generation (RAG) and its variants (Lewis et al., 2020; Yu et al., 2022; Zheng et al., 2023). Han et al. (2024) introduced GraphRAG, which enhances RAG by incorporating graph-structured data, characterized by diverse formats and heterogeneous source.

However, GraphRAG typically assumes access to a pre-constructed, static knowledge graph, and its primary role is to enhance LLM responses by retrieving relevant facts. In contrast, our framework places the LLM agent in an interactive, partially observable environment where critical information is initially unknown. The agent must therefore autonomously collect missing knowledge through exploration, dynamically expanding and updating the Knowledge Graph as new observations are made. This design enables a cyclical reasoning process: the agent leverages the current Knowledge Graph to plan actions, executes those actions to gather new information, and subsequently integrates the acquired knowledge back into the graph. In this way, the Knowledge Graph serves not merely as a retrieval-augmented memory, but as a continually evolving representation that directly supports long-horizon reasoning, adaptive planning, and task completion under uncertainty.

**Addressing Partial Observability via Knowledge Graphs** AriGraph (Anokhin et al., 2024) addresses partial observability by constructing semantic and episodic knowledge graphs. However, it relies on the assumption that the environment provides privileged information, such as global coordinates and a valid action set. This contrasts with our setting, where the agent receives only minimal local observations, such as nearby objects and the main goal. While AriGraph utilizes a hand-coded algorithm to identify unexplored areas based on known locations, our LLM-based agent must autonomously track its position and maintain an internal spatial representation. Furthermore, our framework requires the agent to generate actions across the entire DSL based on the current state, whereas AriGraph selects from an environment-provided action set. This highlights the adaptability of our method to more general applications.

## 7 Conclusion

This paper investigates the application of large language models to navigation in partially observable environments characterized by limited local observations. We focus on equipping LLMs with mechanisms for structured memory, spatial reasoning, and planning under uncertainty. Specifically, we propose a framework that combines domain-specific languages for high-level planning with a dynamically constructed knowledge graph serving as an external memory. Our approach enables the agent to iteratively plan, execute actions, and update information, effectively bridging the gap between abstract logical reasoning and grounded action execution. Experimental results in complex BabyAI environments demonstrate that leveraging a knowledge graph significantly improves planning efficiency, task success rates, and robustness to partial observability. These findings highlight the synergy between LLM-based reasoning and structured, adaptive memory representations, offering a promising paradigm for long-horizon, memory-intensive tasks in dynamic environments through interactions.

## 8 Limitations

While BabyAI effectively represents environments requiring strategic planning under partial observability, it abstracts away the complexities inherent in open-world environments. In environments like Minecraft, for instance, it is difficult to strictly discretize the state space into distinct "nodes." Consequently, extending our framework to such domains would require additional mechanisms for topological mapping or automated state decomposition to define functional areas.

Another limitation lies in the accuracy of LLM-constructed knowledge graphs. As demonstrated in Section 4.3, KG construction errors can cascade into planning failures, particularly when the LLM misclassifies revisited rooms as novel nodes. A lightweight mitigation strategy could leverage trajectory history similarity—matching the current observation to previously seen views or node summaries—to reduce spurious node expansions with minimal additional overhead. We consider this a promising direction for future work.

Finally, when an LLM is used as the downstream actor, action misalignment introduces additional failure modes that further compound KG construction errors. Addressing this through parameter-efficient fine-tuning or finer-grained DSL primitives remains an important avenue for future research.

## Acknowledgment

This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2022-II220311, Development of Goal-Oriented Reinforcement Learning Techniques for Contact-Rich Robotic Manipulation of Everyday Objects (31%), No. RS-2024-00457882, AI Research Hub Project, No. RS-2019-II190079, Artificial Intelligence Graduate School Program (Korea University), the IITP (Institute of Information & Communications Technology Planning & Evaluation)-ITRC (Information Technology Research Center) grant funded by the Korea government (Ministry of Science and ICT) (IITP-2026-RS-2024-00436857) (31%), the NRF (RS-2024-00451162) funded by the Ministry of Science and ICT, Korea, BK21 Four project of the National Research Foundation of Korea, the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2025-00560367), the IITP under the Artificial Intelligence Star Fellowship support program to nurture the best talents (IITP-2026-RS-2025-02304828) grant funded by the Korea government (MSIT) (32%), and KOREA HYDRO & NUCLEAR POWER CO., LTD (No. 2024-Tech-09). The authors would like to thank the Gemini Academic Program Reward for their generous support of this work.

## References

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. 2024. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. *arXiv preprint arXiv:2407.04363*.

Anthropic. 2025. Introducing claude 4. https://www.anthropic.com/news/claude-4.

Shraddha Barke, Emmanuel Anaya Gonzalez, Saketh Ram Kasibatla, Taylor Berg-Kirkpatrick, and Nadia Polikarpova. 2024. Hysynth: Context-free llm approximation for guiding program synthesis. *Advances in Neural Information Processing Systems*, 37:15612–15645.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. 2023. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pages 3676–3713. PMLR.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2019. Babyai: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, volume 105. New Orleans, LA.

Francois Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. 2024. Arc prize 2024: Technical report. *arXiv preprint arXiv:2412.04604*.

Nathan Cloos, Meagan Jens, Michelangelo Naim, Yen-Ling Kuo, Ignacio Cases, Andrei Barbu, and Christopher J Cueva. 2024. Baba is ai: Break the rules to beat the benchmark. *arXiv preprint arXiv:2407.13729*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, and 1 others. 2018. Textworld: A learning environment for text-based games. In *Workshop on Computer Games*, pages 41–75. Springer.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, and 1 others. 2024. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309*.

Zifan He, Yingqi Cao, Zongyue Qin, Neha Prakriya, Yizhou Sun, and Jason Cong. 2025. HMT: Hierarchical memory transformer for efficient long context language processing. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8068–8089, Albuquerque, New Mexico. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, and 1 others. 2021.

Knowledge graphs. *ACM Computing Surveys (Csur)*, 54(4):1–37.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2025. HiAgent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32779–32798, Vienna, Austria. Association for Computational Linguistics.

Adam Tauman Kalai, Ofir Nachum, Santosh S. Vempala, and Edwin Zhang. 2025. Why language models hallucinate. *Preprint*, arXiv:2509.04664.

Myunsoo Kim, Hayeong Lee, Seong-Woong Shim, Junho Seo, and Byung-Jun Lee. 2025. Nbdi: A simple and effective termination condition for skill extraction from task-agnostic demonstrations. In *International Conference on Machine Learning*, pages 30437–30461. PMLR.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. 2020. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684.

Hayeong Lee, Jun Ho Seo, Sunguk Shin, Jinho Lee, Myunsoo Kim, Minsuk Chang, and Byung-Jun Lee. 2025. Evaluating llm planning in partially observable environments via observation representations and action sequences. In *NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Weiyu Ma, Qirui Mi, Yongcheng Zeng, Xue Yan, Runji Lin, Yuqiao Wu, Jun Wang, and Haifeng Zhang. 2024. Large language models play starcraft ii: Benchmarks and a chain of summarization approach. *Advances in Neural Information Processing Systems*, 37:133386–133442.

Drew M. McDermott. 2000. The 1998 ai planning systems competition. *AI Magazine*, 21(2):35.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, and 1 others. 2024. Balrog: Benchmarking agentic llm and vlm reasoning on games. *arXiv preprint arXiv:2411.13543*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. 2021. Minihack the planet: A sandbox for open-ended reinforcement learning research. *arXiv preprint arXiv:2109.13202*.

Alberto Sanfeliu and King-Sun Fu. 2012. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (3):353–362.

Seong-Woong Shim, Myunsoo Kim, Jae Hyeon Cho, and Byung-Jun Lee. 2026. Beyond RAG vs. long-context: Learning distraction-aware retrieval for efficient knowledge grounding. In *The Fourteenth International Conference on Learning Representations*.

SungUk Shin and Youngjoon Kim. 2025. Enhancing graph of thought: Enhancing prompts with llm rationales and dynamic temperature control. In *The Thirteenth International Conference on Learning Representations*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Junjie Wen, Minjie Zhu, Yichen Zhu, Zhibin Tang, Jinming Li, Zhongyi Zhou, Chengmeng Li, Xiaoyu Liu, Yaxin Peng, Chaomin Shen, and 1 others. 2024. Diffusion-vla: Generalizable and interpretable robot foundation model via self-generated reasoning. *arXiv preprint arXiv:2412.03293*.

Jihan Yang, Shusheng Yang, Anjali W. Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. 2025. Thinking in space: How multimodal large language models see, remember, and recall spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10632–10643.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. 2022. Generate rather than retrieve: Large language models are strong context generators. *arXiv preprint arXiv:2209.10063*.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. 2023. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*.

# A Module Detail with Prompt

## A.1 Knowledge Graph-Based Memory

**Verify Node Transition Module** In the *Verify Node Transition*, we prompt a Large Language Model (LLM) with the state-action trajectory and a guiding instruction for reasoning. The LLM analyzes this information to determine the validity of the transition. Recognizing the critical impact of this judgment on the knowledge graph's accuracy, we exclusively employ an ensemble method for this module. The node transition is determined by a majority vote over five trials. To verify a node transition, we query an LLM to check the transition's occurrence and predict the necessary connection information for the subsequent node. To enhance the reliability of the response, we incorporate a self-evaluation mechanism inspired by the Independent Evaluation method (Yao et al., 2023), and augment each module with explicit LLM rationale generation (Shin and Kim, 2025) to improve the transparency and accuracy of decisions.

**Verify Node Transition Prompt**

```
[TEXT INPUT]

{Room_Description}
{room_description}
{/Room_Description}

{Action_Description}
{action_description}
{/Action_Description}

<Trajectories>
<Step_num>
{Trajectory}
{Direction}
</Step_num>

<Action_num>
{action}
</Action_num>

...

</Trajectories>

<Connected_Door_Instruction>
{connected_door_instruction}
</Connected_Door_Instruction>

<Door_Direction_Instruction>
{door_direction_instruction}
</Door_Direction_Instruction>

<Check_Pass_Door_Instruction>
{verify_node_trans_instruction}
</Check_Pass_Door_Instruction>

[IMAGE INPUT]

{images}

-----------------------------------


[OUTPUT]


```LLM Reasoning```
{
  connected_door_rationale,
  connected_door,
  door_direction_rationale,
  door_direction,
  check_pass_door_rationale,
  check_pass_door,
  answer_confidence_score
}
```
```

**Extract Information** In the *Extract Information*, the LLM is prompted to summarize the current observation. It is provided with the observation and guided by instructions for reasoning, analyzing the observation to generate a summary containing only the most critical information. This prompting strategy serves the two purposes of enabling effective differentiation between graph nodes and ensuring efficient memory utilization by storing only essential information. Subsequently, the LLM's responses contain both underscores and spaces, all underscores are converted to spaces for consistent formatting.

**Extract Information Prompt**

```
[TEXT INPUT]

<Observation>
{observation}
</Observation>

<Entity_Listing_Instructions>
{entity_listing_instructions}
</Entity_Listing_Instructions>

<Current_Room_Entities_Instruct>
{current_room_entities_instruct}
</Current_Room_Entities_Instruct>

<Room_Entities_Relate_Instruct>
{room_entities_relate_instruct}
<Room_Entities_Relate_Instruct>

<Direction_Of_Entities_Instruct>
{direction_of_entities_instruct}
</Direction_Of_Entities_Instruct>

[IMAGE INPUT]

{image}

-----------------------------------


[OUTPUT]

```LLM Reasoning```
{
 current_room_entities_rationale,
 current_room_entities,
 entities_relationships_rationale,
 entities_relationships,
 direction_of_entities_rationale,
 direction_of_entities,
}
```
```

**Determine the Current Node** In the *Determine the Current Node*, the LLM determines the current node. It is provided with information about the previously occupied node, the nodes connected to that previous node, and the current observation. The LLM determines whether the current node is a previously visited node or unvisited node, and it responds with the corresponding graph node number. The selection of a graph-based localization method over a coordinate-based approach was driven by the potential for compounding errors when requiring an LLM to manage memory. This memory is intended to mitigate significant error accumulation. The effectiveness of this approach is supported by the GED experiment results. Furthermore, the knowledge graph facilitates the efficient storage of entity information.

**Determine the Current Node Prompt**

```
[TEXT INPUT]

<Observation>
{observation}
</Observation>

<Observed_Entities>
{observed_entities}
</Observed_Entities>

<Room_Information>
- Previous_Room_Number
  {previous_room_number}
<Connected_with_Previous_Rooms>
- The previous room is connected
  with {connection_information}.
- Room contains the entities:
  {entities_information}
</Connected_with_Previous_Rooms>
</Room_Information>

<Rooms_List>
- {nodes}
</Rooms_List>

<Description>
{description}
</Description>

<Current_Graph_Node_Instructions>
{current_graph_node_instructions}
</Current_Graph_Node_Instructions>

-----------------------------------


[OUTPUT]

```LLM Reasoning```
{
  current_graph_node_id_rationale,
  current_graph_node_id
}
```
```

**Update Entity Information** In the *Update Entity Information*, the LLM provides both the graph information and the current observation to synthesize previously observed entity information with the current observation. The environment contains visually identical entities, external information is required to differentiate entities. To enable the LLM to distinguish between these entities, we provided the relational and directional information. The LLM responds with the aggregated observation, including updated relations and directions. Subsequently, we applied a post-processing to convert all underscores in the entity information to spaces.

**Update Entity Information Prompt**

```
[TEXT INPUT]

<Current_Room_Entities>
<Entity_List>
{node_entitiy_list}
</Entity_List>
<Entities_Relationships>
{node_entities_relate_information}
</Entities_Relationships>
<Direction_Of_Entities>
{node_dir_of_ent_information}
</Direction_Of_Entities>
</Current_Room_Entities>

<Partial_Obs_Entity_Information>
<Entity_List>
{entitiy_list}
</Entity_List>
<Entities_Relationships>
{entities_relate_information}
</Entities_Relationships>
<Direction_Of_Entities>
{dir_of_entities_information}
</Direction_Of_Entities>
</Partial_Obs_Entity_Information>

<Door_Change>
{door_change_information}
</Door_Change>

<Inventory_Change>
{inventory_change_information}
</Inventory_Change>

<Instruction>
{instruction}
</Instruction>

-----------------------------------


[OUTPUT]

```LLM Reasoning```
{
  graph_entities_rationale,
  graph_entities,
  graph_entities_relate_rationale,
  graph_entities_relate,
  graph_dir_of_entities_rationale,
  graph_dir_of_entities
}
```
```

### A.2 Planning Using Domain-Specific Language

**Plan with DSLs** In the *Plan with DSLs*, the LLM receives as input the current inventory, the previous plan and its execution status, the current observation, the agent's facing direction, and the set of available DSL instructions with their descriptions. Conditioned on this information, the LLM generates a sequence of DSL instructions, accompanied by a rationale, that aligns with its high-level plan for solving the mission. The generation process leverages the knowledge graph through predefined tool calls invoked by the LLM's decisions. In addition, the LLM specifies a target entity for the plan, together with a rationale, indicating the object on which the current plan should focus.

**Plan with DSLs Prompt**

**[TEXT INPUT]**

```
<Rule_Description>
{rule_description}
</Rule_Description>

<Graph_Information>
{graph_information}
</Graph_Information>

<Subplan_Target_Entity_Instruct>
{subplan_target_entity_instruct}
</Subplan_Target_Entity_Instruct>

<Subplans_Instructions>
{subplans_instructions}
</Subplans_Instructions>

<Inventory>
{inventory}
</Inventory>

<Last_Plan>
{last_plan}
</Last_Plan>

<Last_Plan_Completion>
{last_plan_completion}
</Last_Plan_Completion>

<Facing_Direction>
{facing_direction}
</Facing_Direction>

<DSL_List>
{dsl_list}
</DSL_List>
```

**[IMAGE INPUT]**

```
{image}

-----------------------------------
```

**[OUTPUT]**

```
```LLM Reasoning```
{
  subplan_target_entity_rationale,
  subplan_target_entity,
  subplans_rationale,
  subplans,
}
```
```

**Verify Completion** In the *Verify Completion*, the LLM determines whether the previous plan has been completed and provides a rationale for its judgment. This decision is based on the agent's current information, including its inventory, door traversal status, facing direction, current observation, the previous plan, and the number of times that plan has been repeated. In addition, the LLM evaluates whether the plan should be adjusted—and explains why—if it has remained incomplete for an extended period.

**Verify Completion Prompt**

**[TEXT INPUT]**

```
<Graph_Information>
{graph_information}
</Graph_Information>

<Inventory>
{inventory}
</Inventory>

<Pass_Door>
{pass_door_information}
</Pass_Door>

<Observation>
{observation}
</Observation>

<Facing_Direction>
{facing_direction}
</Facing_Direction>

<Check_DSL_Commands>
{check_DSL_commands}
</Check_DSL_Commands>

<Previous_Plans>
{previous_plans}
</Previous_Plans>

<Last_Plan>
{last_plan}
</Last_Plan>

<Num_Repeats_Last_Plan>
{num_repeats_last_plan}
</Num_Repeats_Last_Plan>

<Is_Complete_Instruction>
{is_complete_instruction}
</Is_Complete_Instruction>

<Need_To_Adjust_Instruction>
{need_to_adjust}
</Need_To_Adjust_Instruction>
```

**[IMAGE INPUT]**

```
{image}

-----------------------------------
```

**[OUTPUT]**

```
```LLM Reasoning```
{
  is_complete_rationale,
  is_complete,
  need_to_adjust_rationale,
  need_to_adjust,
}
```
```

**Execute** In the *Execute*, the LLM-as-agent, acting as the actor, analyzes the mission, the current observation, and its facing direction, and generates up to 10 low-level actions in a single turn, accompanied by a rationale. The instruction prompt supplies the LLM-as-agent with the available action set, the transition dynamics of the environment, and a concise guideline on how to handle blockers when encountered.

**Execute Prompt**

**[TEXT INPUT]**

```
<Rule_Description>
{rule_description}
</Rule_Description>

<Action_Description>
{action_description}
</Action_Description>

<Mission_Description>
{mission_description}
</Mission_Description>

<Graph_Information>
{graph_information}
</Graph_Information>

<Mission>
{subplan}
</Mission>

<Observation>
{problem}
</Observation>

<Direction>
You are facing north.
</Direction>

<Inventory>
{inventory}
</Inventory>

<Instructions>
{instructions}
</Instructions>
```

**[IMAGE INPUT]**

```
{image}

-----------------------------------
```

**[OUTPUT]**

```
```LLM Reasoning```
{
    actions_rationale,
    actions
}
```
```

### A.3 Tool Call

To accommodate this structure, we implemented three retrieval tools that provide the LLM with action sequences and trajectories to summarize (i) the most recent decision, (ii) historical information about a queried entity, and (iii) historical information about the most recently observed closed or locked door.

**Get Neighbor Entity Information** Designed for short-term planning, it operates by receiving a node number as a parameter to return a string with all information about the specified node and its neighbors. Since the LLM cannot natively determine if all nodes have been visited, the tool also provides a visitation count for each node within the current decision step to inform the agent's exploration strategy.

**Search Closest Entity** It receives a node and a target entity as parameters, performs a Breadth-First Search (BFS), and provides information on the nearest node containing that entity. If the entity is not present in memory, it returns information that the entity has not been discovered. Conversely, if the nearest entity is found, it provides the sequence of node transitions required to reach it. This information helps the agent determine whether it needs to perform further exploration or formulate a long-term plan.

**Find Unexplored Closed Door** It is designed to find the shortest path to the nearest closed or locked door from a given node. It receives the node as a parameter and performs a Breadth-First Search (BFS). If no such door is found in the memory, it returns a notification that the closed or locked door is undiscovered. Otherwise, it returns the sequence of node transitions that constitutes the path to the nearest closed or locked door. This information enables the agent to formulate long term exploration plans.

## B Environment Details

We extend BabyAI (Chevalier-Boisvert et al., 2019), a partially observable 2D gridworld simulation. Built on the MiniGrid platform, BabyAI supports efficient simulation and offers a range of instruction-following tasks using a simplified synthetic language called Baby Language. Each layout consists of $n$ rooms connected by colored doors, with objects placed throughout. Objects are defined by color and type. While unlocked doors can be opened freely, locked doors require keys of the matching color. At each time step, the agent receives a partial observation representing its $7 \times 7$ field of view. Walls and doors obstruct the observation, even when doors are open.

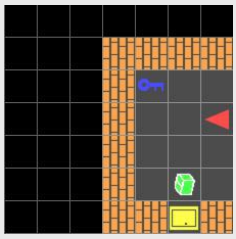
The environment provides observations in two modalities: pixel-based images and textual descriptions. While BabyAI offers default image-rendered assets, we modify the object assets in the pixel-based observations to enhance visual distinctiveness and improve object recognition by LLMs. The textual representation encodes each cell using predefined object descriptors (e.g., Wall, Yellow_Closed_Door, Blue_Box), separated by semicolons, enabling precise symbolic reasoning over the observed grid. This structured format enables symbolic reasoning over spatial configurations while preserving compatibility with language-based models. The action space supports six actions: `Go Forward`, `Turn Left`, `Turn Right`, `Pickup`, `Drop`, and `Toggle`. The `Toggle` action allows the agent to interact with doors, such as opening, closing, or unlocking them.

In our experimental setup, we focus on two challenging missions, `OpenDoor` and `PutNextTo`. In `OpenDoor`, the agent must retrieve a key and unlock a corresponding door located elsewhere in the layout. To guarantee that tasks require exploration, we randomly generate diverse layouts and missions, filtering them to ensure that key entities are placed in non-adjacent rooms. In `PutNextTo`, the agent must find two distinct objects placed in separate rooms and bring them together, making success contingent on navigating multiple rooms rather than exploiting local information.

### B.1 Experimental Setup: Layout Generation

The environments follow grid layouts of $2 \times 2$ and $3 \times 3$ with complexity further increased by including at least one locked door in each layout. To effectively evaluate exploration efficiency, we fil-

Image Observation



Text Observation

```
Unseen;Unseen;Unseen;Unseen;Unseen;  Unseen;              Unseen;
Unseen;Unseen;Unseen;Wall;  Wall;    Wall;                Wall;
Unseen;Unseen;Unseen;Wall;  Blue_Key;Empty;               Empty;
Unseen;Unseen;Unseen;Wall;  Empty;   Empty;               Agent;
Unseen;Unseen;Unseen;Wall;  Empty;   Empty;               Empty;
Unseen;Unseen;Unseen;Wall;  Empty;   Green_Box;           Empty;
Unseen;Unseen;Unseen;Wall;  Wall;    Yellow_Closed_Door;Wall;
```

Figure 7: Two observation formats: a pixel-based image and a textual description.

ter layouts to guarantee that accessing the target objects requires obtaining a key to unlock a door. In particular, for $3 \times 3$ layouts, we enforce that completing the mission necessitates exploring at least four rooms. To ensure environmental diversity, we generated five random layouts under two object-density conditions: one object per room and three entities per room. In total, our experiments cover 20 unique layouts.

## C Comparison of Computational Costs

We compare the token-level costs of different memory structures across our method (dynamic KG) and the baselines: stacked memory (SM), which uses a linear, stack-structured alternative, and static memory (static KG), where the entire knowledge graph is provided in the context window. We are expecting that our method improves the agent's performance with the minimal cost. We present the token-level costs associated with different memory structures. The experiments were conducted using Gemini-2.5-Flash models as the planner, with heuristic bots acting as the downstream actors. The values reported below represent the number of tokens consumed per decision step, averaged over five independent runs in Table 5. Additionally, the model generates the following number of output tokens per decision step in Table 6.

As Stacked Memory (SM) does not utilize a KG, costs associated with KG-specific prompt are not applicable (denoted as "-"). "Verify Node Transition" module utilizes an ensemble of LLMs. The reported cost assumes a single LLM; total cost scales linearly with the ensemble size.

In our experiment, the average number of decision steps required by our framework was 33.33 for $2 \times 2$ layouts and 46.1 for $3 \times 3$ layouts. In contrast, the SM baseline required 46.93 and 70.43 steps, respectively. This demonstrates a key trade-off: while the per-step cost of our method is slightly higher due to the overhead of managing the KG, **the total trajectory cost** remains comparable. This is evident in more complex problems ($3 \times 3$ layouts), where our method's ability to solve tasks in fewer steps offsets the higher per-step token consumption.

## D Ablation Study on Observation Modalities

Table 7 shows the experimental results comparing performance when the agent receives environmental information as text-only versus when image observations are also provided. When using only the text observation, the agent succeeded in 38 out of 60 trials. In contrast, when image observations were added, the agent succeeded in 49 out of 60 trials. This suggests the LLM achieves a better understanding of the environment, as it can leverage the additional information from the images.

Table 5: The number of tokens consumed per decision step.

| Task | Ours | SM | Static KG |
|---|---|---|---|
| Verify Node Transition | 3761.96 | - | 3949.38 |
| Determine the Current Node | 286.32 | - | 359.01 |
| Extract Information | 1557.91 | 1579.28 | 1571.01 |
| Plan with DSLs | 2354.95 | 2188.42 | 720.98 |
| Verify Completion | 1212.96 | 1148.71 | 1248.21 |
| Update Entity Information | 1613.03 | - | 1707.11 |
| Total Tokens | 10787.13 | 4916.41 | 9555.70 |
| Total Costs ($) | 0.003 | 0.001 | 0.003 |

Table 6: The number of output tokens per decision step.

| Task | Ours | SM | Static KG |
|---|---|---|---|
| Verify Node Transition | 1687.30 | - | 1745.88 |
| Determine the Current Node | 67.68 | - | 79.17 |
| Extract Information | 491.34 | 509.95 | 490.93 |
| Plan with DSLs | 102.49 | 124.79 | 90.36 |
| Verify Completion | 160.85 | 179.35 | 162.79 |
| Update Entity Information | 764.07 | - | 845.10 |
| Total Tokens | 3273.83 | 814.09 | 3414.23 |
| Total Costs ($) | 0.008 | 0.002 | 0.009 |

Table 7: Performance comparison between text-only and text with image on PutNextTo missions. We evaluate the performance of the knowledge graph approach using two different observation.

| Observation Format | # rooms | Accuracy (%) |
|---|---|---|
| Image & Text | 2×2 | 96.7 ± 3.3 |
| | 3×3 | 66.7 ± 8.6 |
| Text | 2×2 | 83.3 ± 6.6 |
| | 3×3 | 43.3 ± 9.0 |

Table 8: Success rate of `PutNextTo` mission. We compare dynamic memory, where the agent controls the tool calls, and static memory, where all information is always provided.

| # rooms | Metrics | Gemini-2.5-Flash | |
|---|---|---|---|
| | | Dynamic Memory | Static Memory |
| 2×2 | Success (%) | 96.7 ± 3.3 | 93.3 ± 4.8 |
| | GED | 4.50 ± 1.65 | 2.79 ± 1.09 |
| 3×3 | Success (%) | 66.7 ± 8.6 | 58.6 ± 9.03 |
| | GED | 7.80 ± 1.63 | 9.38 ± 2.17 |

## E Example Case

We visualize the subplans produced by the LLM planner augmented with a LLM-generated knowledge graph, along with the full execution trajectory of a downstream actor, in Figure 8 and Figure 9.

We further highlight the failure modes of LLM planner augmented with stacked memory, where every attempt fails to complete the mission, as shown in Figure 10 and Figure 11.

## F Comparison of Memory Access Methods

To evaluate the LLM's ability to handle the dynamics of information gathering and utilizing in sequential decision-making, we compare two settings: **dynamic memory** where the model performs tool-calling experiments, and **static memory**, where the entire knowledge graph is provided in the context window.

The results are summarized in Table 8. Although static memory provides the LLM with more information at each step, its performance was lower. This finding is consistent with the comparison between knowledge graph and stacked memory, suggesting that inefficient memory structures can hinder the performance.

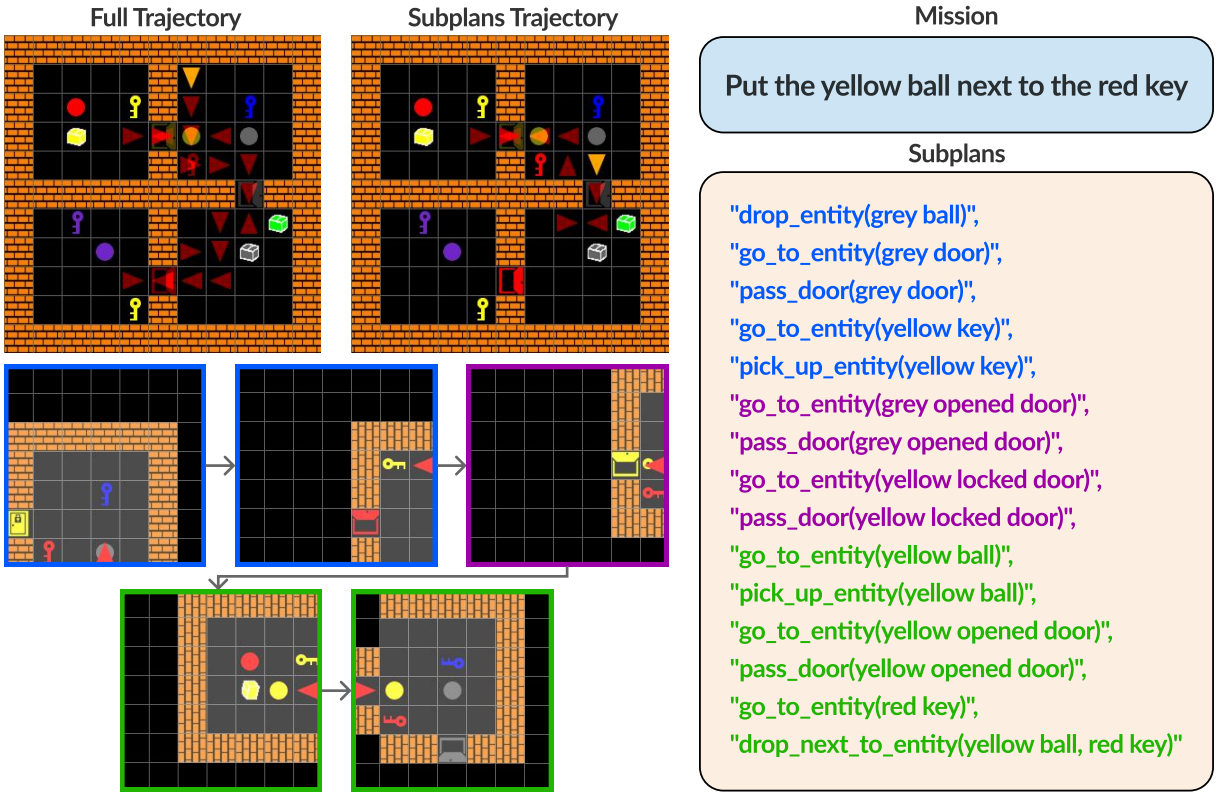



Figure 8: The LLM planner augmented with a knowledge graph successfully generates a long, coherent sequence of subplans to accomplish the mission: “Put the yellow ball next to the red key." The agent moves from the start position (▸) to the final position (▸). The LLM generated entire sequence of subplans at once, demonstrating its capabilities for long-horizon reasoning in partially observable environments.

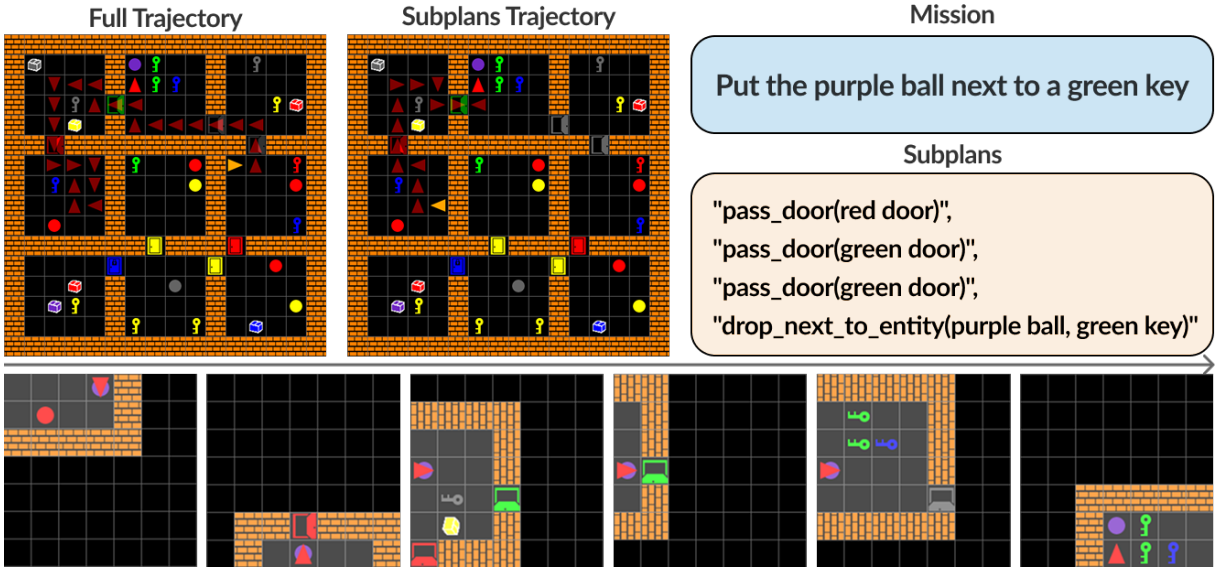



Figure 9: The LLM planner, using a knowledge graph, creates a long and logical series of subplans to complete the mission: "Put the purple ball next to a green key." The agent moves from the start position (▸) to the final position (▸).

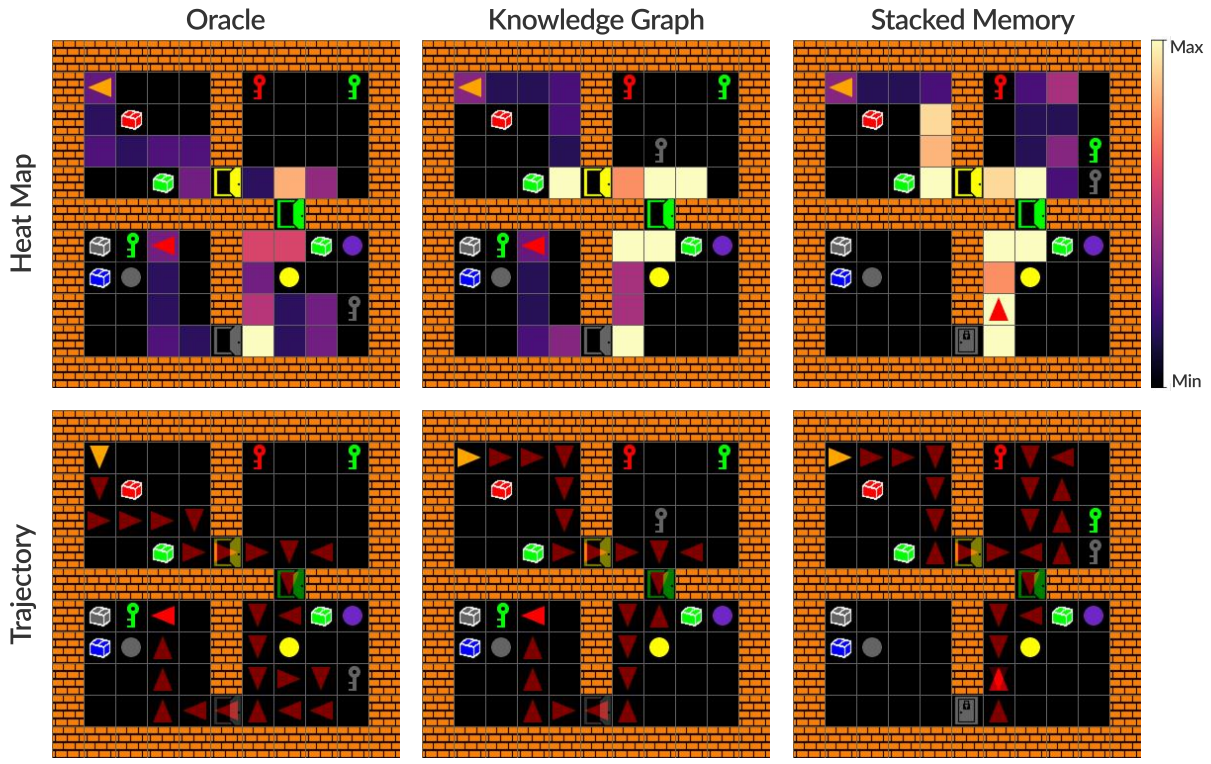



Figure 10: The visitation heat map and the trajectory of oracle agent, knowledge graph-augmented agent, and stacked memory-augmented agent.

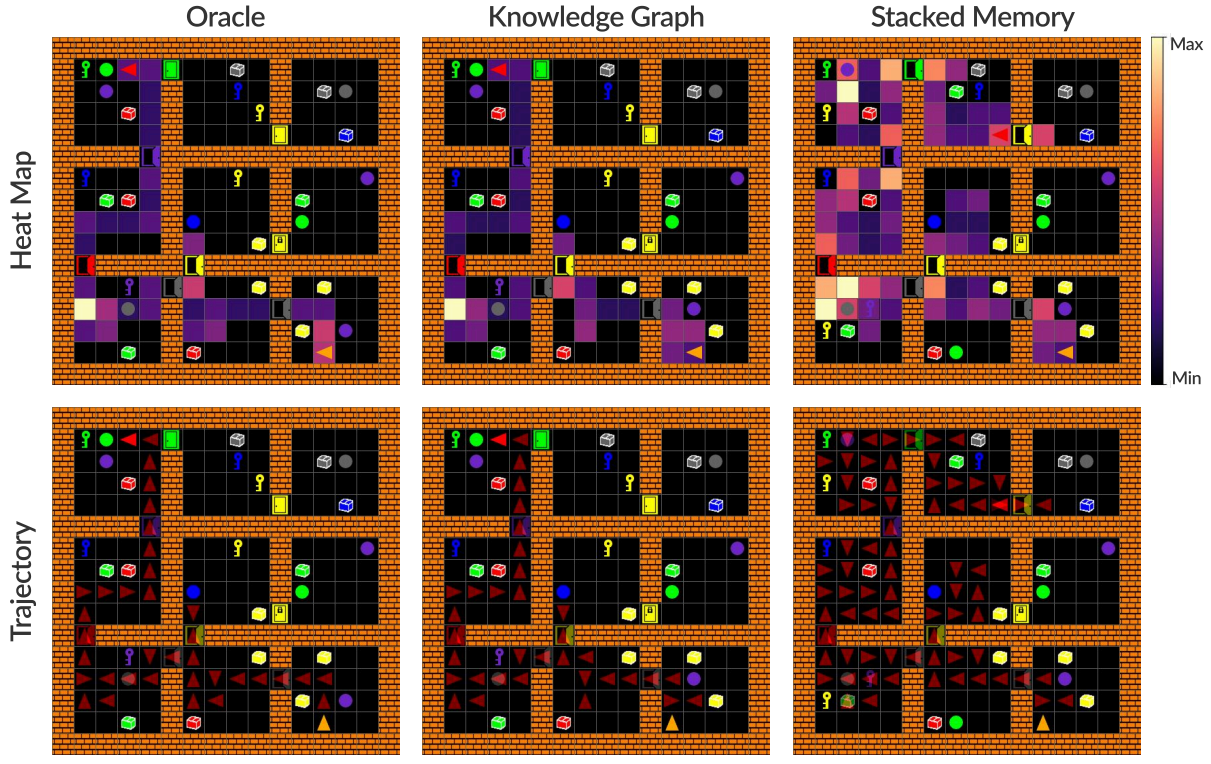



Figure 11: The visitation heat map and the trajectory of oracle agent, knowledge graph-augmented agent, and stacked memory-augmented agent.