SPATIALAGENT: WORKING MEMORY FOR LONG-HORIZON SPATIAL REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon interactive tasks require large language model (LLM) agents to retain and use information gathered across many steps, particularly in partially observed environments where spatial knowledge remains relevant throughout an episode. However, existing working-memory approaches are not generally organized around the spatial structure of the environment itself. We investigate whether explicitly maintaining a structured spatial representation in memory can improve both agent performance and inference efficiency. We introduce *SpatialAgent*, an agent framework that constructs a persistent map from textual observations and provides it to an underlying LLM. The map is adapted to the structure of each environment and updated incrementally throughout the episode as new information is observed. We evaluate SpatialAgent across multiple environments from the BALROG benchmark, comparing it with existing agent baselines while also studying model scale, alternative map-construction methods, and inference efficiency. Results show that SpatialAgent significantly outperforms existing agent baselines across most evaluated models and environments. We further find that SpatialAgent generally improves over an otherwise equivalent no-map variant, indicating that the explicit spatial representation itself contributes to performance; SpatialAgent can reduce inference cost by maintaining strong performance with shorter context windows and less expensive models. These results suggest that spatially structured working memory can be a useful design choice for LLM agents operating in long-horizon environments while reducing reliance on extensive trajectory context.

1 INTRODUCTION

As LLM-based agent research increasingly moves toward tasks that require sustained interaction over long horizons, these agents must operate effectively across extended sequences of observations and actions (Paglieri et al., 2025). In these environments, completing a task may depend on information observed many steps earlier. For spatially relevant tasks, this includes information about objects, entities, and previously explored regions that may be needed for decisions later in the episode (Zhang et al., 2024). Because the agent typically receives only a partial description of the environment at each step, this spatial knowledge becomes distributed across a long sequence of observations. A common approach is to retain the trajectory history, often together with model reasoning, in the LLM's context (Yao et al., 2023). For long-horizon tasks where the context becomes very large, effectively maintaining consistent, relevant, and up-to-date spatial information in working memory is therefore a central challenge. In this paper, we use interactive text games as a testbed and investigate the challenges of partial observability, navigation, exploration, and reasoning over spatial-memory, which also arise in real-world embodied environments (Shah et al., 2022; Rajvanshi et al., 2024).

Existing working-memory approaches provide alternatives to retaining a growing interaction trajectory, including structured representations that organize accumulated information into explicit entities and relationships (Gutiérrez et al., 2025; Chhikara et al., 2025; Anokhin et al., 2025; Rasmussen et al., 2025; Xu et al., 2025). However, most are not designed specifically around preserving the spatial organization of long-horizon environments. See Table 1. AriGraph Anokhin et al. (2025) is one exception: it constructs a knowledge graph from textual observations that includes spatial relationships for TextWorld-style environments. It retrieves relevant facts from the knowledge graph and

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

lists them as text in the LLM's context. SpatialAgent instead provides a compact map that mirrors the spatial organization surrounding the agent. Spatial information such as distance, directions, and navigation paths accumulated throughout an episode is directly available on the map for reasoning, and the map representation is generalizable across a variety of environments. This choice parallels findings from cognitive psychology showing that humans construct spatial mental models from textual descriptions, abstracting spatial relationships from the sequence in which the information was presented (Tversky, 1991; Taylor & Tversky, 1992). In this paper, we investigate whether LLMs can effectively navigate and reason over such representations, as humans do, and whether doing so improves planning and decision making in long-horizon interactive tasks.

		Memory			
Agent	**Domain**	**Content**	**Representation**	**Construct**	**Spatial Context**
ReAct (Yao et al., 2023)	General	Interaction trajectory	Thought action observation history	Append	No
Generative Agents (Park et al., 2023)	Social simulation	Events, dialogue, plans, reflections	Memory stream + reflections	LLM + heuristic	No
DEPS (Wang et al., 2024b)	Minecraft / ALFWorld / robotics	State, failures, plans	State descriptions + plans	LLM + algorithm	No
Voyager (Wang et al., 2023)	Minecraft	Skills + execution feedback	Executable JavaScript skill library	LLM	No
GITM (Zhu et al., 2023)	Minecraft	Action sequences + game knowledge	Reference-plan memory	LLM	No
MemGPT (Packer et al., 2024)	Dialogue / document QA	History, facts, summaries, documents	Hierarchical main / recall / archival memory	LLM-directed	No
HiAgent (Hu et al., 2024b)	Planning / interactive tasks	Subgoal histories + completed-subgoal summaries	Hierarchical subgoal-indexed text	LLM	No
StateAct (Rozanov & Rei, 2025)	Interactive environments	Goal, state, reasoning, actions	Chain-of-states structured text	LLM	No
AriGraph (Anokhin et al., 2025)	Text games / QA	Entities, relations, locations, actions, episodes	Semantic + episodic knowledge graph	LLM	Text
SpatialAgent (Ours)	Text games	Known/unknown locations, relations, properties	Map	Algorithm, LLM	Map

Table 1: Memory mechanisms in representative LLM agents (details and references in Section 4).

We introduce SpatialAgent, an agent that constructs and maintains a structured spatial map from observations encountered throughout an episode and provides this representation to an underlying LLM during planning and action selection. Rather than requiring the model to repeatedly recover persistent environmental state from its interaction trajectory, the map maintains structured spatial information across steps while a limited amount of recent trajectory context captures short-term interaction details. We study this approach in four text-based interactive environments from BALROG Paglieri et al. (2025): BabyAI (Chevalier-Boisvert et al., 2019), Crafter (Hafner, 2022), BabaIsAI (Cloos et al., 2025), and TextWorld (Côté et al., 2019). These environments collectively require navigation, exploration, object interaction, resource collection, rule-based reasoning and planning. We evaluate SpatialAgent against baselines that rely on trajectory-based context and other working-memory representations. We also examine how map utility varies with model size, compare alternative map-construction methods, and study the performance–cost trade-off and inference efficiency.

Our experiments show that SpatialAgent significantly outperform baselines across most models and environments, demonstrating that LLMs are able to use and benefit from the map representation. SpatialAgent also exhibits scaling behavior: bigger models can generally benefit more from the map representation, and even Claude Sonnet 5 greatly improved performance after using SpatialAgent. Surprisingly, we found that in some environments, LLM-generated Python code can construct and update maps better than human-written code. We also show SpatialAgent achieved strong performance with low inference cost. Our main contributions are:

- We introduce a persistent map-based working-memory representation for long-horizon text environments and study how effectively LLMs of different sizes can reason over it.
- We develop three approaches for constructing spatial maps from textual observations: (1) human-written code, (2) LLM-generated code, and (3) direct LLM extraction, and compared their effectiveness in our evaluation.

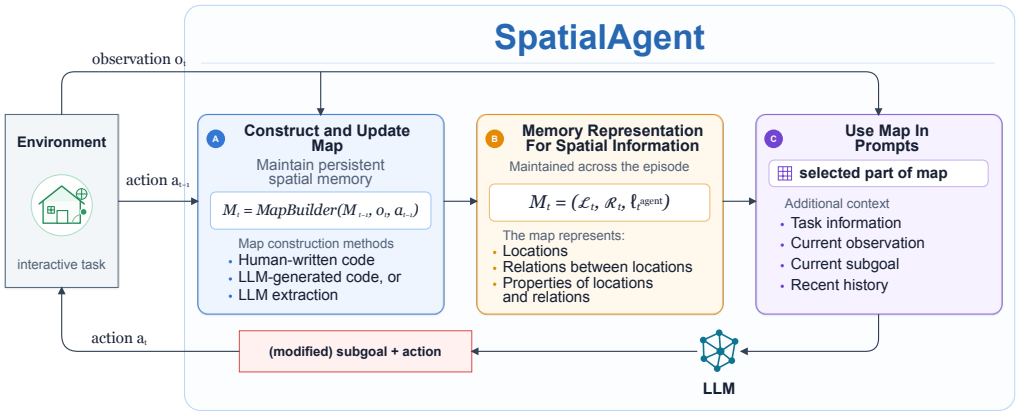


Figure 1: Overview of SpatialAgent. The MapBuilder maintains a persistent spatial map from observations and actions, which is provided to the LLM for decision making.

- We introduce *SpatialAgent*, which integrates map construction, update and usage in LLM planning and action selection in an agent loop for game environments; we experimentally demonstrate that SpatialAgent significantly outperforms baselines in most models and environments, including frontier model Claude Sonnet 5.

2 METHOD

2.1 SPATIALAGENT OVERVIEW

We introduce **SpatialAgent**, an agent architecture that maintains an explicit map of the environment as persistent working memory for long-horizon decision making. Figure 1 provides an overview of the architecture. At each interaction step t, SpatialAgent receives a textual observation o_t from the environment together with the previously executed action a_{t-1}. An environment-specific **MapBuilder** uses this information to update the agent's persistent map M_{t-1}:

$$M_t = \text{MapBuilder}(M_{t-1}, o_t, a_{t-1}). \tag{1}$$

The map M_t stores the agent's accumulated spatial knowledge using the common representation defined in Section 2.2. Each MapBuilder implements the environment-specific logic required to interpret observations and actions and translate them into updates to this representation. The resulting map is then rendered and incorporated into the LLM's decision context as described in Section 2.5.

2.2 SPATIAL MEMORY REPRESENTATION

At interaction step t, SpatialAgent represents its persistent spatial knowledge as a map

$$M_t = \left(\mathcal{L}_t, \mathcal{R}_t, \ell_t^{\text{agent}}\right) \tag{2}$$

where $\mathcal{L}_t$ is the set of identified locations, $\mathcal{R}_t$ contains known spatial relationships between locations, and $\ell_t^{\text{agent}} \in \mathcal{L}_t$ denotes the agent's current location.

A *location* is an environment-specific spatial unit over which spatial relationships can be defined. The type of location may therefore vary across environments without changing the underlying map representation. For example, locations correspond to individual cells in Crafter, BabyAI, and BabaIsAI, whereas in TextWorld a location corresponds to a discrete area such as a room, supermarket, or street. Each location $\ell_i \in \mathcal{L}_t$ may additionally contain a set of environment-specific properties $p_i^L \in 2^{\mathcal{A}_L}$, where $\mathcal{A}_L$ denotes the set of possible location properties. These properties can include terrain, objects, entities, obstacles, or other spatially grounded information. For example, a Crafter

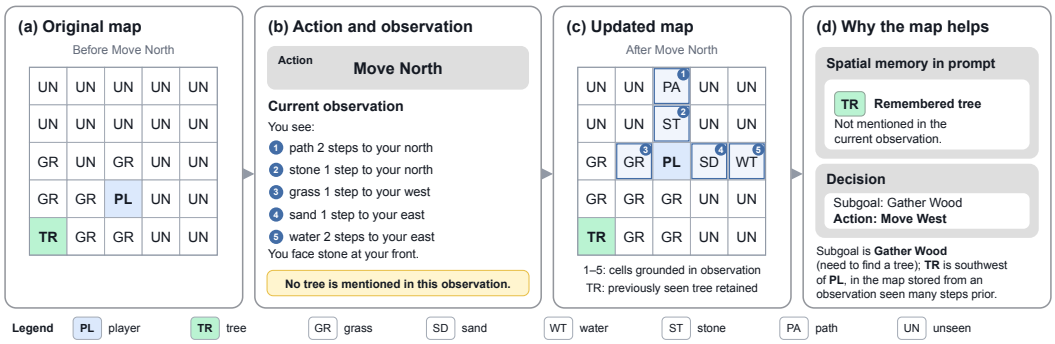


Figure 2: SpatialAgent's persistent spatial memory in Crafter.

location may contain both grass terrain and a plant occupying that terrain, while a TextWorld location may contain a sofa.

We define the set of spatial relationships as

$$\mathcal{R}_t \subseteq \mathcal{L}_t \times (\mathcal{L}_t \cup \{\text{UN}\}) \times 2^{\mathcal{A}_R} \tag{3}$$

where $\mathcal{A}_R$ denotes the set of possible relationship properties. A relationship $(\ell_i, \ell_j, p_{ij}^R) \in \mathcal{R}_t$ indicates that locations ℓ_i and ℓ_j are spatially related, with $p_{ij}^R \in 2^{\mathcal{A}_R}$ containing the known properties of that relationship. Relationship properties can include direction as well as other spatial information such as connectors and their states. For example, in TextWorld, a driveway may be *east* of a living room and connected to it by an *open front door*; both are represented as properties in p_{ij}^R.

The resulting map represents the agent's current spatial knowledge from the existing explorations/episodes, rather than a complete description of the environment. Spatial information observed steps ago remains in the map unless it is updated or invalidated by subsequent observations or environment-specific dynamics.

Figures 2(a,c) and 3 illustrate this shared representation in Crafter and TextWorld, respectively. In Crafter, cells serve as locations with terrain, resource, and entity properties, whereas TextWorld uses larger named areas with properties attached to both locations and their relationships. Thus, the same representation supports environments with different spatial granularity.

The special value UN denotes a location that is known to exist through a spatial relationship but whose identity or location properties have not yet been observed. In Figure 3, an unexplored northward exit from a bedroom can be represented by a relationship whose destination is UN and whose relationship properties include *north*. Once the destination is discovered, UN can be replaced by the corresponding identified location while retaining or updating the properties of the relationship.

2.3 Map Construction and Updating

SpatialAgent constructs its map incrementally throughout an episode. Based on the previous action a_{t-1} and the resulting observation o_t, the environment-specific MapBuilder may identify locations, add or revise location properties, establish or revise spatial relationships and their properties, or update the agent's current location.

Figure 2(a–c) illustrates this process in Crafter. Panel (a) shows part of the map before the current update, including terrain and resources observed during earlier steps. The agent then executes *Move*

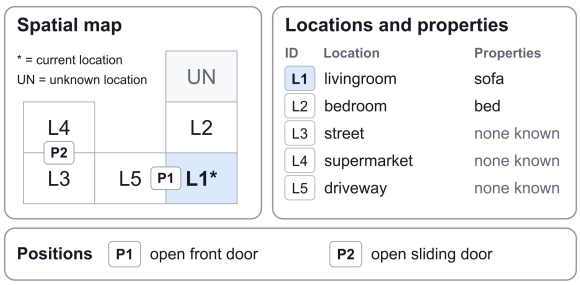


Figure 3: Our map representation for TextWorld.

North and receives the observation shown
in panel (b), which describes newly visible surroundings relative to the agent's new location. The MapBuilder updates the agent's location based on the executed movement and resulting observation, then grounds each relative spatial description into the corresponding map location (see the numbered annotations in panels (b) and (c)). Newly observed properties are incorporated into the corresponding locations, and spatial relationships are established according to their relative directions, producing M_t in panel (c). In our default setting, these updates are performed using deterministic, environment-specific rules written by the authors, requiring no additional LLM calls.

2.4 Alternative Map Construction Methods

As shown in Figure 1, we also evaluate two LLM-based alternative approaches for map construction. Across all three variants, the target map representation and its role within SpatialAgent remain fixed; only the mechanism used to construct and update the map changes.

LLM-generated Python code. We use an LLM to generate a deterministic MapBuilder (Python code) from a short bootstrap trajectory consisting of the initial observation and the first four action–observation transitions. The generation prompt (provided in the appendix) includes this trajectory together with an environment-specific specification of the spatial information to maintain, the required map representation, and the expected MapBuilder behavior, but does not provide the human-written parsing or update rules. The generated code has the same responsibilities as the human-written MapBuilder described in section 2.3: interpreting observations and actions, updating the agent's location, and adding or revising the corresponding locations, relationships, and properties in the map. For example, the Crafter prompt instructs the LLM to generate Python code that maintains the agent's position and observed terrain, resources, and entities using the prescribed map format. The LLM generates the MapBuilder code from the bootstrap trajectory. The generated MapBuilder is then applied to that trajectory to reconstruct the current map state and is used deterministically for the remainder of the episode.

Direct LLM extraction. We prompt an LLM to directly construct and update the map at each step when new action is performed. The prompt (provided in the appendix) includes the existing map, previous action, current observation, and an environment-specific specification defining the spatial information to maintain and the required output format. The LLM returns the complete updated map while preserving previously observed information that remains valid. Unlike LLM-generated code, this approach does not produce a reusable update procedure and therefore requires an additional LLM call at every interaction step.

2.5 Map-Augmented Decision Context

To select the next subgoal and action, SpatialAgent renders a relevant portion of the persistent map M_t as text and adds it to the LLM's decision prompt. The complete map remains in memory; the rendering determines which spatial information is exposed to the LLM at the current step. As shown in Figure 1(c), the map is provided alongside the task information, current observation and subgoal, and recent trajectory history. SpatialAgent then makes a single call to the underlying LLM using this combined context, which jointly produces an updated subgoal and the next environment action.

Figure 2(d) illustrates how our spatial map can influence this decision. Although the current observation in panel (b) does not mention a tree, the updated map in panel (c) retains the tree (`TR`) observed earlier in the episode. When this map is included in the decision context, the LLM can use the tree's stored location to determine that moving west will progress the current subgoal *Gather wood*, leading it to select the action *Move West*. This allows previously observed spatial information to influence decisions even after it leaves the recent trajectory.

The map is rendered using compact environment-specific tokens and a corresponding legend to limit the amount of spatial context added to the prompt. The amount of map context exposed to the LLM is configurable. In Crafter, BabyAI, and BabaIsAI, SpatialAgent renders a local two-dimensional window centered on the agent; in TextWorld, it includes the current location and discovered locations within a fixed spatial distance.

3 EXPERIMENT

In this section, we investigate the following research questions:

- **RQ1:** How does SpatialAgent compare with existing non-spatial working-memory approaches on long-horizon tasks?
- **RQ2:** How does model size affect an LLM agent's ability to benefit from spatial maps provided as context?
- **RQ3:** How do different spatial map construction methods—human-written code, LLM-generated code, and direct LLM extraction—affect agent performance?
- **RQ4:** How does SpatialAgent trade off task performance and input-token cost across different models?

3.1 EXPERIMENTAL SETUP

Benchmark. We evaluate our approach using BALROG (Paglieri et al., 2025), a benchmark for evaluating language model agents on long-horizon interactive game environments. We evaluate all tasks from four text-only BALROG environments: BabyAI, Crafter, BabaIsAI, and TextWorld. Together, these environments cover navigation, object interaction, exploration, resource collection, and rule-based reasoning. For TextWorld, we evaluate all three tasks—Coin Collector, Treasure Hunter, and The Cooking Game—and report their average performance.

Evaluation Metrics. Following BALROG, we report normalized progress on a 0–100 scale. BabyAI and BabaIsAI use binary task-completion scores, while Crafter and TextWorld provide partial credit for intermediate objectives. We report mean score using BALROG's default evaluation episode counts, with a single attempt per episode.

Models. Following BALROG's broad model coverage, we evaluate 8 open-weight and proprietary language models from OpenAI, Anthropic, Meta, and Qwen, including *llama* and *qwen* family models, *gpt-oss-20b* as well as a frontier Claude model (*sonnet-5*). All models are evaluated without fine-tuning or parameter updates; full model and generation configurations are in Appendix A.

3.2 RQ1: COMPARISON WITH AGENT BASELINES

To compare SpatialAgent with existing working-memory approaches, we evaluate against ReAct (Yao et al., 2023), HiAgent (Hu et al., 2024b), and StateAct (Rozanov & Rei, 2025). The three baselines represent trajectory-based reasoning, hierarchical working memory, and explicit textual state tracking, respectively. We adapt HiAgent and StateAct to BALROG while following their original formulations as closely as possible. We also attempted to run AriGraph Anokhin et al. (2025) as a baseline; however, its implementation is highly customized for TextWorld, making it difficult to faithfully adapt their method to other game environments. AriGraph and BALROG include a different set of tasks from TextWorld, so we can not directly compare the results. Due to the high cost for running game evaluations through the Claude API, we limit Claude experiments to two randomly selected games, *Crafter* and *BabyAI*, and compare SpatialAgent against the stronger results from the base Claude agent and Claude with ReAct. We focus on ReAct because it is the strongest baseline in the majority of the open-source model–environment comparisons.

Results. Table 2 shows that SpatialAgent significantly outperforms the non-spatial baselines for most model–environment combinations. On Claude Sonnet 5, SpatialAgent improves over its best baseline, ReAct, in all evaluated environments, improving performance from 82.0 to 90.0 on BabyAI and from 55.5 to 60.5 on Crafter. This suggests that stronger models can effectively exploit the spatial representation.

For open-source models, SpatialAgent shows many notable gains. On BabyAI with Qwen3-4B, performance increases from 28.0 for the strongest baseline to 72.0, while on BabaIsAI with gpt-oss-20b, it increases from 28.5 to 52.5. Interestingly, we observed that for one of the most challenging environments, Crafter, smaller models do not benefit from our map until Qwen3-8B. Using Qwen3-14B and gpt-oss-20b, SpatialAgent achieves the strongest performance among the compared methods in

	BabyAI				Crafter				BabaIsAI				TextWorld			
Model	**ReAct**	**HiAgent**	**StateAct**	**Ours**	**ReAct**	**HiAgent**	**StateAct**	**Ours**	**ReAct**	**HiAgent**	**StateAct**	**Ours**	**ReAct**	**HiAgent**	**StateAct**	**Ours**
llama-3.2-3B-it	22.4	19.2	1.6	**30.0**	**22.3**	11.8	4.5	21.4	8.0	6.5	10.0	**18.3**	1.5	0.5	0.4	**2.4**
qwen3-4B	14.4	24.8	28.0	**72.0**	7.7	**12.3**	7.7	8.6	18.0	10.5	24.0	**25.0**	2.3	0.0	0.0	**5.1**
llama-3.1-8B-it	41.6	28.8	10.4	**46.0**	**25.0**	4.5	14.6	20.5	**25.0**	6.5	12.5	20.0	2.8	0.0	1.9	**5.9**
qwen3-8B	5.6	**40.0**	4.8	36.0	8.2	5.5	7.7	**21.4**	19.5	**21.5**	**21.5**	20.0	**4.2**	0.6	1.6	1.2
qwen3-14B	24.8	52.0	40.8	**76.0**	13.2	9.5	10.9	**27.7**	31.5	11.5	26.5	**43.3**	7.3	1.2	3.2	**21.4**
gpt-oss-20b	27.2	72.8	64.8	**74.5**	**28.6**	16.4	18.6	**28.6**	21.0	19.0	28.5	**52.5**	2.0	0.2	0.5	**6.1**
qwen3-32B	16.0	**70.4**	42.4	62.0	13.2	13.2	15.9	**28.2**	31.5	24.0	**47.5**	30.0	24.2	2.5	16.1	**28.2**
claude-sonnet-5	82.0	–	–	**90.0**	55.5	–	–	**60.5**	–	–	–	–	–	–	–	–

Table 2: Performance of SpatialAgent, ReAct, HiAgent, and StateAct across BALROG environments. Results are reported as mean normalized progress score. The best result for each model and environment is shown in bold.

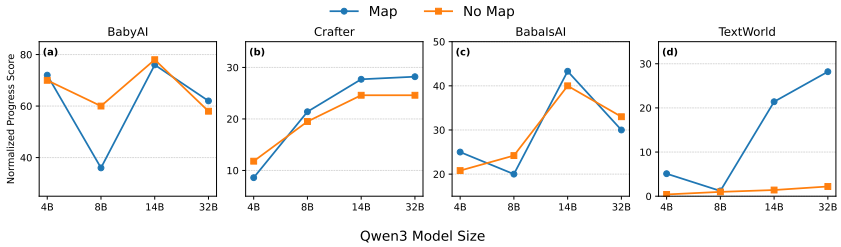


Figure 4: Performance across Qwen3 model family *with* and *without* map two configurations on BALROG environments.

all four environments. Overall, SpatialAgent provides substantial improvements across a range of models and environments.

3.3 RQ2: Effect of Model Size on Understanding the Maps

Table 2 reveals a trend where explicit spatial memory becomes more beneficial as model sizes increase, while the relative improvements of the other baselines remain more consistent across model sizes. For example, for TextWorld, HiAgent reports poor performance across all model sizes; for BabyAI, ReAct did not show a very obvious improvement over larger models. This motivates a more controlled analysis of how the benefit of explicit spatial memory changes as model size increases. We conduct an ablation using Qwen3-4B, 8B, 14B, and 32B. At each model size, we compare SpatialAgent with an otherwise equivalent no-map variant that uses the same decision-making prompt but omits the spatial map from context. By comparing SpatialAgent with a no-map variant, we isolate the contribution of explicit spatial memory while varying model size. This allows us to examine whether larger models can interpret and exploit the map representation better than smaller models.

Results. In Figure 4, Crafter exhibits the clearest scaling trend: at the smaller sizes, SpatialAgent does not benefit from the map, but as the model sizes increase, and the map becomes useful and the SpatialAgent outperforms its no-map variant at 8B, 14B, and 32B. TextWorld shows an even larger separation at higher model sizes: spatial context provides little benefit at 8B but improves performance by 20.0 points at 14B and 26.0 points at 32B. Interestingly, this pattern is consistent with the scaling behavior observed in Table 2. It also suggests that the benefit of spatial information depends partially on the environment. In environments such as Crafter and TextWorld, where successful behavior can require retaining spatial information across many locations collected through dispersed observations, larger models can better understand complex maps and thus may be better able to make use of them. In smaller environments like BabaIsAI, the agent may rely more on spatial information in the local observation, reducing the benefit of reasoning over the added map. Overall, the results suggest that increased model capacity can improve the ability to exploit spatial maps in some environments, particularly Crafter and TextWorld, but model size alone does not determine whether spatial context is beneficial.

Model	Environment	Human-written	LLM-generated	Direct LLM
Qwen3-14B	BabyAI	68.0	**76.0**	68.0
Qwen3-14B	Crafter	**27.7**	20.9	0.0
Qwen3-14B	BabaIsAI	33.3	**43.3**	0.0
Qwen3-14B	TextWorld	**21.4**	5.3	1.0
gpt-oss-20B	BabyAI	**74.5**	70.0	58.0
gpt-oss-20B	Crafter	**28.6**	27.3	0.0
gpt-oss-20B	BabaIsAI	**52.5**	48.3	0.0
gpt-oss-20B	TextWorld	**6.1**	2.8	1.37

Table 3: Performance of the three spatial map construction methods for the top SpatialAgent models from RQ1. Results are reported as mean normalized progress score.

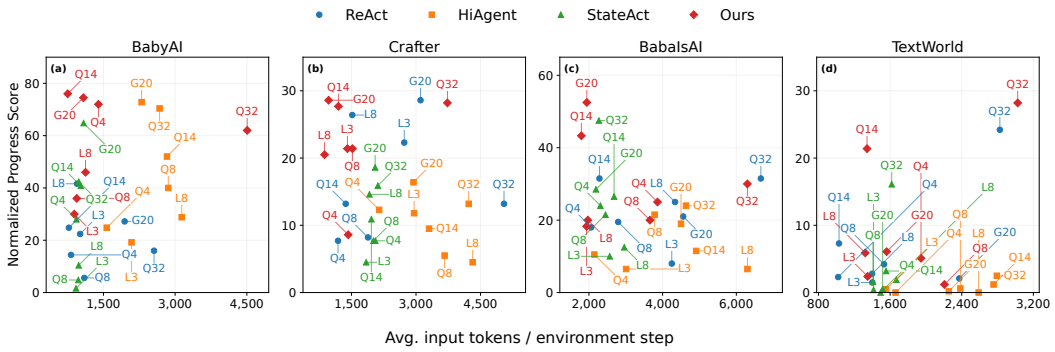


Figure 5: Perform-cost comparison: Llama 3.2-3B (L3), Llama 3.1-8B (L8), Qwen3 models (Q4/Q8/Q14/Q32), and gpt-oss-20B (G20). X-axis shows inference efficiency, Y-axis lists game scores.

3.4 RQ3: Spatial Map Construction

We compare the three map-construction methods described in Section 2.4: human-written code, LLM-generated code, and direct LLM extraction. To isolate the effect of the map-construction method, all variants target the same spatial representation and use the same SpatialAgent decision-making prompt. We conduct this comparison using two larger open-source models, Qwen3-14B and gpt-oss-20b, across all four BALROG environments. The prompts used for the two LLM-based construction methods are provided in the appendix.

Results. Table 3 shows that LLM-generated Python code performs strongly across most settings, remaining competitive with human-written code on BabyAI, Crafter, and BabaIsAI and even outperforming it in some cases. With Qwen3-14B, LLM-generated code improves over human-written code on both BabyAI and BabaIsAI, while with gpt-oss-20b it remains close to human-written performance across all three of these environments. Its performance degrades substantially on TextWorld. Direct LLM extraction faces significant challenges in the more difficult environments. These results indicate that directly reconstructing the map through repeated LLM inference is substantially less reliable than maintaining spatial state through a deterministic update program.

3.5 RQ4: Inference Efficiency and Performance–Cost Trade-offs

To evaluate inference efficiency, we use mean input tokens per step, following prior work (Chen et al., 2025). We first measure this cost for every agent–model combination reported in Table 2. For each environment, we plot task performance against mean input tokens per step, with points corresponding to the different models evaluated for each agent.

Results. SpatialAgent achieves a favorable balance between the performance and toke efficiency: its configurations frequently attain the highest or near-highest performance (red dots on the top)

while remaining at relatively low or moderate token costs (red dots on the left). Many red dots on the top left corners indicate that there are always some models where SpatialAgent achieved closet to top scores yet remains token efficient for all four environments. We also observed that StateAct is a token efficient model and HiAgent is costly while ReAct's token efficiency degrades as model sizes increase.

4 Related Work

Recent work has explored memory mechanisms that compress, encode, or structure past experience within an agent's working context. Summarization-based methods (Packer et al., 2024; Chhikara et al., 2025; Zhou et al., 2025; Yu et al., 2025; Zhong et al., 2023) condense interaction histories into natural-language memories, while latent-memory methods (Wang et al., 2024a; 2025; Zhang et al., 2025) encode experience into compact internal representations (Hu et al., 2026). Structured approaches organize task information explicitly through symbolic or relational representations (Chhikara et al., 2025; Edge et al., 2025; Rasmussen et al., 2025; Sun & Zeng, 2025; Xu et al., 2025). HiAgent (Hu et al., 2024b) hierarchically manages working memory around subgoals, while StateAct (Rozanov & Rei, 2025) maintains an evolving chain of states for long-horizon interaction. AriGraph (Anokhin et al., 2025) is particularly related, constructing a semantic and episodic knowledge graph while exploring text environments. Its implementation is designed around its graph-based agent pipeline rather than the unified multi-environment setting considered in our evaluation.

Recent research on spatial reasoning in large language models has mostly focused on multimodal settings, where models reason over images and videos (Zhao et al., 2026; Yang et al., 2025; Mao et al., 2025; Shiri et al., 2024). A smaller body of work studies textual spatial reasoning, where spatial relations or environments are represented through natural language. Chain-of-Symbol (Hu et al., 2024a) compresses spatial information into symbolic states. Visualization-of-Thought (Wu et al., 2024) and neuro-symbolic approaches (Premsri & Kordjamshidi, 2025) introduce intermediate representations or formal constraints to support spatial reasoning. These agents are not required to explore an environment and retain crucial information obtained over observations. General-purpose agent frameworks such as ReAct, Reflexion, and AdaPlanner (Yao et al., 2023; Shinn et al., 2023; Sun et al., 2023) have been evaluated in text-based embodied environments such as ALFWorld (Shridhar et al., 2021). However, in these approaches, locations and objects are stored implicitly in reasoning traces rather than explicitly modeled.

To contextualize our approach within prior work on memory for LLM-based agents, Table 1 compares representative agent-memory approaches in terms of memory content, representation, construction, and explicit spatial structure. In contrast, SpatialAgent maintains an explicit spatial representation as working memory that is incrementally updated and used throughout an episode.

5 Conclusions

In this work, we introduce SpatialAgent, an LLM-based agent designed to maintain spatial information throughout long-horizon text interactions. SpatialAgent incrementally updates an explicit representation of the environment from textual observations and includes a relevant portion of this map in the LLM prompt to support decision making. We evaluate SpatialAgent across four BALROG environments and multiple language models from Anthropic, OpenAI, Meta, and Qwen, comparing it with non-spatial working-memory baselines to analyze model scale, map-construction strategies, and inference efficiency. SpatialAgent frequently improves task performance, with the best improvements scaling up to larger models and more spatially complex environments. We also find that LLM-generated MapBuilders can sometimes outperform human-written implementations, while direct LLM-based map updates are less reliable. Finally, explicit spatial memory can reduce dependence on long trajectory histories and improve the trade-off between performance and input-token cost. Our results support explicit spatial memory as a useful mechanism for long-horizon tasks that involve reasoning about spatial information, even for Claude Sonnet 5.

AI USE STATEMENT

Our work designs an agent framework, which has components of using AI to generate Python code to build maps and using LLM to directly build maps (Section 2.4). We used AI tools to correct grammatical errors in writing, and construct diagrams in the paper. Additionally, we used generative AI tools to assist with creating and editing a limited portion of the experimental code, primarily for plot generation and baseline environment setup. The research ideas, methodological decisions, experimental design, interpretation of results, and scientific conclusions were developed by the human authors. We did not use generative AI tools for other tasks requiring disclosure under the ICLR 2027 AI Policy for Authors.

We have reviewed all AI-assisted work. The authors manually inspected and executed the AI-generated code, reviewed language edits to preserve the intended meaning, and checked the diagrams for consistency with the underlying methods and concepts. We take responsibility for the final content of this work, including all text, code, figures, claims, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To support reproducibility, we will release our implementation as open-source. When the OpenReview discussion forum opens, we will provide reviewers and area chairs with a link to an anonymous repository containing all implementation code for this work.

REFERENCES

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. Arigraph: Learning knowledge graph world models with episodic memory for llm agents, 2025. URL https://arxiv.org/abs/2407.04363.

Sirui Chen, Mengshi Zhao, Lei Xu, Yuying Zhao, Beier Zhu, Hanwang Zhang, Shengjie Zhao, and Chaochao Lu. Depo: Dual-efficiency preference optimization for llm agents, 2025. URL https://arxiv.org/abs/2511.15392.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning, 2019. URL https://arxiv.org/abs/1810.08272.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Nathan Cloos, Meagan Jens, Michelangelo Naim, Yen-Ling Kuo, Ignacio Cases, Andrei Barbu, and Christopher J. Cueva. Baba is ai: Break the rules to beat the benchmark, 2025. URL https://arxiv.org/abs/2407.13729.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games, 2019. URL https://arxiv.org/abs/1806.11532.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization, 2025. URL https://arxiv.org/abs/2404.16130.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models, 2025. URL https://arxiv.org/abs/2405.14831.

Danijar Hafner. Benchmarking the spectrum of agent capabilities, 2022. URL https://arxiv.org/abs/2109.06780.

Hanxu Hu, Hongyuan Lu, Huajian Zhang, Yun-Ze Song, Wai Lam, and Yue Zhang. Chain-of-symbol prompting elicits planning in large langauge models, 2024a. URL https://arxiv.org/abs/2305.10276.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model, 2024b. URL https://arxiv.org/abs/2408.09559.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents, 2026. URL https://arxiv.org/abs/2512.13564.

Yongsen Mao, Junhao Zhong, Chuan Fang, Jia Zheng, Rui Tang, Hao Zhu, Ping Tan, and Zihan Zhou. Spatiallm: Training large language models for structured indoor modeling, 2025. URL https://arxiv.org/abs/2506.07491.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. Balrog: Benchmarking agentic llm and vlm reasoning on games, 2025. URL https://arxiv.org/abs/2411.13543.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL https://arxiv.org/abs/2304.03442.

Tanawan Premsri and Parisa Kordjamshidi. Neuro-symbolic training for reasoning over spatial language. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2395–2414. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.128. URL http://dx.doi.org/10.18653/v1/2025.findings-naacl.128.

Abhinav Rajvanshi, Karan Sikka, Xiao Lin, Bhoram Lee, Han-Pang Chiu, and Alvaro Velasquez. Saynav: Grounding large language models for dynamic planning to navigation in new environments, 2024. URL https://arxiv.org/abs/2309.04077.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory, 2025. URL https://arxiv.org/abs/2501.13956.

Nikolai Rozanov and Marek Rei. Stateact: Enhancing llm base agents via self-prompting and state-tracking, 2025. URL https://arxiv.org/abs/2410.02810.

Dhruv Shah, Blazej Osinski, Brian Ichter, and Sergey Levine. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action, 2022. URL https://arxiv.org/abs/2207.04429.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Fatemeh Shiri, Xiao-Yu Guo, Mona Golestan Far, Xin Yu, Gholamreza Haffari, and Yuan-Fang Li. An empirical analysis on spatial reasoning capabilities of large multimodal models, 2024. URL https://arxiv.org/abs/2411.06048.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning, 2021. URL https://arxiv.org/abs/2010.03768.

Haoran Sun and Shaoning Zeng. Hierarchical memory for high-efficiency long-term reasoning in llm agents, 2025. URL https://arxiv.org/abs/2507.22925.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models, 2023. URL https://arxiv.org/abs/2305.16653.

Holly A Taylor and Barbara Tversky. Spatial mental models derived from survey and route descriptions. *Journal of Memory and Language*, 31(2):261–292, 1992. ISSN 0749-596X. doi: https://doi.org/10.1016/0749-596X(92)90014-O. URL https://www.sciencedirect.com/science/article/pii/0749596X92900140.

Barbara Tversky. Spatial mental models. volume 27 of *Psychology of Learning and Motivation*, pp. 109–145. Academic Press, 1991. doi: https://doi.org/10.1016/S0079-7421(08)60122-X. URL https://www.sciencedirect.com/science/article/pii/S007974210860122X.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. Memoryllm: Towards self-updatable large language models, 2024a. URL https://arxiv.org/abs/2402.04624.

Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogerio Feris, and Zexue He. M+: Extending memoryllm with scalable long-term memory, 2025. URL https://arxiv.org/abs/2502.00592.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents, 2024b. URL https://arxiv.org/abs/2302.01560.

Wenshan Wu, Shaoguang Mao, Yadong Zhang, Yan Xia, Li Dong, Lei Cui, and Furu Wei. Mind's eye of llms: Visualization-of-thought elicits spatial reasoning in large language models, 2024. URL https://arxiv.org/abs/2404.03622.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

Jihan Yang, Shusheng Yang, Anjali W. Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. Thinking in space: How multimodal large language models see, remember, and recall spaces, 2025. URL https://arxiv.org/abs/2412.14171.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent, 2025. URL https://arxiv.org/abs/2507.02259.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents, 2025. URL https://arxiv.org/abs/2509.24704.

Yue Zhang, Ziqiao Ma, Jialu Li, Yanyuan Qiao, Zun Wang, Joyce Chai, Qi Wu, Mohit Bansal, and Parisa Kordjamshidi. Vision-and-language navigation today and tomorrow: A survey in the era of foundation models, 2024. URL https://arxiv.org/abs/2407.07035.

Haoyu Zhao, Akide Liu, Zeyu Zhang, Weijie Wang, Feng Chen, Ruihan Zhu, Gholamreza Haffari, and Bohan Zhuang. Cov: Chain-of-view prompting for spatial reasoning, 2026. URL https://arxiv.org/abs/2601.05172.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory, 2023. URL https://arxiv.org/abs/2305.10250.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory, 2023. URL https://arxiv.org/abs/2305.17144.

A Model configuration and generation.

We used `meta-llama/Llama-3.2-3B-Instruct`, `meta-llama/Llama-3.1-8B-Instruct`, `Qwen/Qwen3-4B`, `Qwen/Qwen3-8B`, `Qwen/Qwen3-14B`, `Qwen/Qwen3-32B`, and `openai/gpt-oss-20b`, served through vLLM. Our main ReAct experiments used temperature $T = 1.0$ and a maximum generation length of 8192 tokens per request. Model-native thinking was disabled for Qwen3 using `enable_thinking=false`, while GPT-OSS used low reasoning effort. Hosted experiments used Anthropic's `claude-sonnet-5` and `claude-haiku-4-5-20251001`. Sonnet5 used adaptive thinking with low effort and used an 8,192-token output ceiling in the experiments. Temperature was omitted for Anthropic requests, leaving the provider default. Other sampling parameters were not overridden from original BALROG setup.

B Additional Tables and Results

Table 4: Performance across BALROG environments. Values are mean normalized progress (%) $\pm$ standard error. TextWorld combines its three games.

	BabyAI			**Crafter**			**Baba Is AI**			**TextWorld**		
Model	ReAct	HiAgent	StateAct	ReAct	HiAgent	StateAct	ReAct	HiAgent	StateAct	ReAct	HiAgent	StateAct
Llama 3.2-3B	22.4 ± 3.7	19.2 ± 3.5	1.6 ± 1.12	22.3 ± 2.4	11.8 ± 1.6	4.54 ± 1.11	8.0 ± 1.9	6.5 ± 1.7	10.0 ± 2.12	1.47 ± 0.64	0.49 ± 0.21	0.39 ± 0.27
Qwen3-4B	14.4 ± 3.1	24.8 ± 3.9	28.0 ± 4.02	7.7 ± 2.23	12.3 ± 2.4	7.73 ± 2.65	18.0 ± 2.7	10.5 ± 2.2	24.0 ± 3.12	2.25 ± 0.68	0.00 ± 0.00	0.00 ± 0.00
Llama 3.1-8B	41.6 ± 4.4	28.8 ± 4.1	10.4 ± 2.73	25.00 ± 2.25	4.5 ± 1.1	14.55 ± 2.47	25.0 ± 2.2	6.5 ± 1.7	12.5 ± 2.34	2.84 ± 0.92	0.00 ± 0.00	1.86 ± 0.67
Qwen3-8B	5.6 ± 2.0	40.0 ± 4.4	4.8 ± 1.91	8.2 ± 2.0	5.45 ± 1.5	7.73 ± 2.04	19.5 ± 2.8	21.5 ± 2.9	21.5 ± 2.90	4.22 ± 1.75	0.59 ± 0.41	1.57 ± 0.53
Qwen3-14B	24.8 ± 3.9	52.0 ± 4.5	40.8 ± 4.40	13.2 ± 2.7	9.5 ± 1.8	10.91 ± 2.25	31.5 ± 3.3	11.5 ± 2.3	26.5 ± 3.12	7.25 ± 2.52	1.18 ± 0.50	3.24 ± 1.13
GPT-OSS-20B	27.2 ± 4.0	72.8 ± 4.0	64.8 ± 4.27	28.6 ± 4.6	16.4 ± 4.3	18.64 ± 3.55	21.0 ± 2.9	19.0 ± 2.8	28.50 ± 3.19	1.96 ± 0.66	0.20 ± 0.14	0.49 ± 0.25
Qwen3-32B	16.00 ± 3.28	70.4 ± 4.1	42.4 ± 4.42	13.2 ± 2.0	13.2 ± 2.3	15.91 ± 2.75	31.50 ± 3.28	24.0 ± 3.0	47.5 ± 4.6	24.22 ± 4.65	2.45 ± 0.68	16.08 ± 3.67
Claude Sonnet 5	82.00 ± 5.49	–	–	55.45 ± 4.00	–	–	–	–	–	–	–	–

C Prompts for Alternative Map Construction

The following prompts were used for the two LLM-based map-construction methods described in Section 2.4

C.1 LLM-GENERATED PYTHON CODE

Prompt used to generate the environment-specific MapBuilder. The spatial representation specification and initial interaction trajectory are inserted at runtime.

```
Write a deterministic Python MapBuilder that maintains persistent
spatial memory for an agent interacting with an environment.

The MapBuilder will be generated once from an initial interaction
trajectory and then used without any additional LLM calls for the
remainder of the episode.

Implement exactly one class named:

GeneratedMapBuilder

It must subclass BaseMapBuilder:

class GeneratedMapBuilder(BaseMapBuilder):

BaseMapBuilder and the Python re module are already available in the
execution environment. Do not include import statements.

The class must implement:

- reset()
- update(obs, prev_action=None)
- get_map_str(...)
- get_legend_str()
- get_prompt_instructions()

Requirements:
- The implementation must be deterministic.
- Do not call an LLM, API, network service, or external process.
- Do not read or write files.
- Use only the provided observation and previous action to update the
  persistent spatial state.
- Maintain spatial information across calls to update().
- Do not store plans, reasoning, subgoals, or task solutions unless they
  are explicitly part of the spatial representation specification.
- Follow the required spatial representation and output format exactly.
- Do not use import statements.
- The BaseMapBuilder class and re module are already available.

ENVIRONMENT AND SPATIAL REPRESENTATION SPECIFICATION:

<ENVIRONMENT-SPECIFIC SPATIAL REPRESENTATION SPECIFICATION>

INITIAL INTERACTION TRAJECTORY:

<INITIAL INTERACTION TRAJECTORY>

Use the interaction trajectory to infer how observations and actions
relate to spatial-state updates.

Return only valid Python source code defining GeneratedMapBuilder.
Do not include Markdown code fences or explanations.
```

C.2 DIRECT LLM EXTRACTION

Prompt used to update the spatial memory at each interaction step. The environment-specific format, current spatial memory, previous action, and current observation are inserted at runtime.

```
You maintain a persistent spatial memory for an agent interacting with an
     environment.

Your only task is to update the spatial memory. Do not choose an action,
    plan, or solve the task.

Follow these rules:
- Preserve previously established spatial information unless the new
    observation contradicts it.
- Add or change information only when supported by the current
    observation.
- Do not invent unseen locations, objects, or spatial relationships.
- Do not remove remembered information simply because it is absent from
    the current observation.
- A previous movement action may have failed; do not assume movement
    succeeded unless supported by the observation.
- Return the complete updated spatial memory.
- Return only the spatial memory, with no explanation and no Markdown
    code fences.

ENVIRONMENT-SPECIFIC SPATIAL MEMORY FORMAT:
<ENVIRONMENT-SPECIFIC SPATIAL MEMORY FORMAT>

CURRENT SPATIAL MEMORY:
<CURRENT SPATIAL MEMORY>

PREVIOUS ACTION:
<PREVIOUS ACTION>

CURRENT OBSERVATION:
<CURRENT OBSERVATION>

UPDATED SPATIAL MEMORY:
```