What Do LLM Agents Know About Their World? Task2Quiz: A Paradigm for Studying Environment Understanding

Anonymous ACL submission

Abstract

Large language model (LLM) agents have demonstrated remarkable capabilities in complex decision-making and tool-use tasks, yet their ability to generalize across varying environments remains a under-examined concern. Current evaluation paradigms predominantly rely on trajectory-based metrics that measure task success, while failing to assess whether agents possess a grounded, transferable model of the environment. To address this gap, we propose Task-to-Quiz (T2Q), a deterministic and automated evaluation paradigm designed to decouple task execution from world-state understanding. We instantiate this paradigm in T2QBench, a suite comprising 30 environments and 1,967 grounded QA pairs across multiple difficulty levels. Our extensive experiments reveal that task success is often a poor proxy for environment understanding, and that current memory machanism can not effectively help agents acquire a grounded model of the environment. These findings identify proactive exploration and fine-grained state representation as primary bottlenecks, offering a robust foundation for developing more generalizable autonomous agents.

1 Introduction

Large language model (LLM) agents have recently shown impressive competence in realistic, multi-step decision making and tool use (Yehudai et al., 2025; Luo et al., 2025; Matarazzo and Torlone, 2025; Minaee et al., 2025), enabling progress in web navigation (Wei et al., 2025), software engineering (Zhang et al., 2024; Dong et al., 2025), and operating-system interaction (Hu et al., 2025). While, the issue of generalization is concerned by the community. Agent capabilities that look strong on particular scenarios often fail to transfer reliably when the environment changes or constraints are slightly modified(Liu et al., 2025). This gap raises a basic but under-examined question: do LLM agents actually acquire a grounded model of the environment, or do they primarily learn specific heuristics that optimize task metrics?

In this paper, we aim at investigating agent's environment understanding beyond current task success evaluation. As shown in Figure 1, most existing agent benchmarks are trajectory-based. They ask whether an agent reaches the goal, such as success rate or the quality of intermediate actions, measuring "doing". However, generalization or transferability requires "knowing". An agent may complete a task while still lacking robust world-state knowledge, such as where objects are, how rooms connect, which direction relationships hold, or what latent object states (e.g., locked/closed) are. Conversely, an agent may acquire correct environment facts yet fail due to planning errors or long-horizon constraints. Therefore, we highlight environment-based evaluation to provide diagnostic "knowing" signals, support controlled interventions (e.g., varying layout complexity or reachability constraints), and enables reproducible grading grounded in known world metadata rather than human annotation or stochastic LLM-as-judge scoring. However, there are three challenges: (i) how to encourage agents to explore beyond single-goal exploitation; (ii) how to quantify environment understanding in a fine-grained, comparable way; and (iii) how to build a fully automated and reproducible pipeline without manual labeling or judge hallucinations (Kalai et al., 2025).

To address the issues, we propose **Task-to-Quiz (T2Q)**, an automated environment-based evaluation paradigm. We instantiate T2Q on TextWorld-style text games, where the full environment metadata (topology, entity placements, and symbolic state relations) is available by construction, making verification precise and convenient. There are two stages. Stage 1 synthesizes coverage-oriented task sets that drive agents to traverse rooms, interact with objects, and reveal stateful

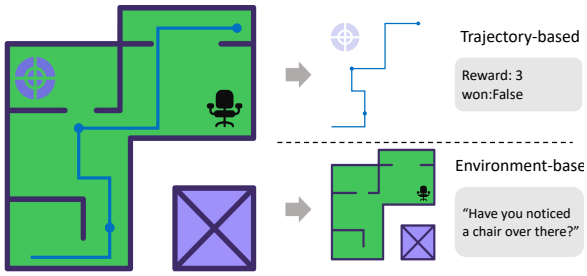

Figure 1: Comparison of trajectory-based and environment-based Evaluation, the trajectory-based evaluation cares about whether and how the final goal is reached, while the environment-based evaluation cares about the entire understanding of the environment.

constraints; task generation is designed to increase exposure to diverse environment elements rather than only optimizing a single end goal. Stage 2 converts the environment and the executed trajectories into a multi-dimensional quiz that probes complementary aspects of world-state knowledge, including localization, connectivity, direction, key-lock matching, and latent properties. Importantly, each question is paired with trajectory-based prerequisites: if an agent's interactions could not have revealed the necessary evidence (e.g., it never visited a room or never opened a container), the question is marked non-answerable to avoid penalizing agents for information they had no opportunity to observe. Because answers are computed by a verifier grounded in environment metadata and interaction logs, the entire evaluation is deterministic and reproducible. This design separates failures of exploration/exposure from failures of state tracking and reasoning, enabling a fair and diagnostic measure of world-state knowledge.

Using this paradigm, we build T2QBench, a controlled evaluation suite spanning three difficulty levels with 30 environments, 224 coverage-oriented tasks, and 1,967 environment-grounded QA pairs. We evaluate diverse proprietary and open-source models under a unified protocol that jointly measures two complementary dimensions: Task Success Rate (TSR) and Environment Understanding Score (EUS), capturing post-interaction world-state knowledge. Across extensive experiments with different memory mechanisms, we obtain three consistent insights. First, task success is indeed not a reliable proxy for environment understanding. TSR and EUS can diverge as difficulty increases. Second, existing memory systems do not consistently improve environment understanding; in many cases, a naive in-context baseline matches or outperforms memory-augmented agents, suggesting that current memory pipelines may lose fine-grained evidence. Third, a low propensity for proactive exploration is a dominant bottleneck. Agents perform relatively better on questions answerable via short-term recall, but struggle on aspects that require actively uncovering latent properties or relations. Together, these results suggest that improving generalization may require not only better planning and retrieval, but also mechanisms that explicitly support world-state acquisition and representation.

Our contributes can be summarized as follows:

- We propose a deterministic environment-based evaluation paradigm **Task-to-Quiz (T2Q)**, that separates "doing" from "knowing". T2Q contains two stages: 1) coverage-oriented task generation and 2) quiz evaluation, whose answer is dynamically generated based on the agent's interaction history and the environment metadata.

- We construct a fully automated pipeline and build **T2QBench**, enabling reproducible measurement on different environments, tasks, and difficulty levels.

- We conduct an empirical investigation across models and memory systems that reveals key bottlenecks in agents' environment understanding ability.

2 Controllable Environment Construction

In this section, we describe our controllable environment construction pipeline. We first present the base environment generation procedure, and then introduce the task coverage planning pipeline.

2.1 Environment Generation

Layout and Connectivity Generation We build the base game environments on top of TextWorld Framework (Côté et al., 2019) with a controllable configuration space. Specifically, we procedurally generate room layouts and connectivity and instantiate an explicit number of interactive objects. As a result, the full environment metadata is known by construction (e.g., connectivity, locations, and object attributes) for future verification and question generation.

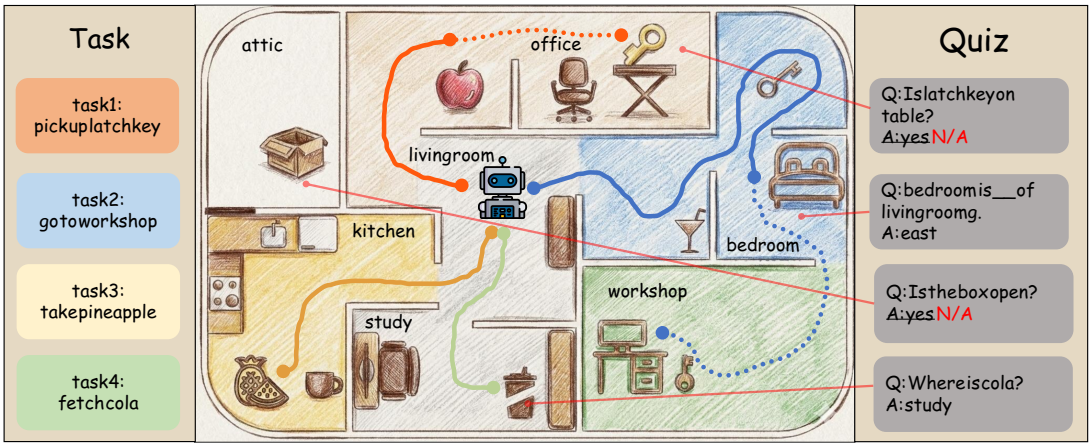

Figure 2: Overview of our T2Q paradigm. It contains coverage-oriented tasks, and environment-based quizzes that are used to evaluate the agent's environment understanding. The answer of each question is dynamically generated based on the agent's interaction history and the environment metadata.

Reachability Design The default TextWorld generation pipeline provides limited control over reachability because locks and keys are often sampled randomly, which may lead to weak coupling between access constraints and traversal structure. To make reachability controllable and systematically challenging, we add a post-generation refinement step that injects lock–key dependencies while preserving solvability: we first compute the reachable region from the start, then lock a fixed fraction of doors/containers (0.4 at our default setting) to enforce multi-step, long-horizon planning.

Distractor Placement To further increase reasoning difficulty without altering overall solvability, distractors are added during world construction. These include irrelevant objects and keys that are interactable but useless for task completion.

2.2 Task Coverage Planning

Modeling Task Generation as Weighted Set Cover Problem To comprehensively evaluate an agent's environment understanding capability, we aim to maximize its opportunities to interact with the environment. Since the exploration stage is task-driven, we design a task set that covers as much of the reachable area and interactive entities as possible while minimizing redundancy among tasks. Therefore, task generation is formulated as a **weighted set cover problem** (Algorithm 1). First, all reachable rooms and interactive entities are extracted from the environment instance $\mathcal{G}$ to form the target universe $\mathcal{R}$. They are considered as the targets that final task set should cover. For each target in $\mathcal{R}$, a corresponding target-specific candidate task is instantiated. To accomplish each task goal, the agent must execute a sequence of actions; we refer to the corresponding reference solution trajectory as a *walkthrough*. Accordingly, a winning walkthrough is derived via TextWorld task analysis function using available metadata $\mathcal{K}$, producing a candidate task set $\mathcal{S}$. These generated tasks serve as the candidate pool for the subsequent coverage selection step.

Algorithm 1 Coverage Task Generation

Require: Environment instance $\mathcal{G}$, metadata/knowledge $\mathcal{K}$
Ensure: Greedy task set $\mathcal{Q}$

$\mathcal{R} \leftarrow \textsc{ExtractTargets}(\mathcal{G})$ ▷ Universe of targets
$\mathcal{S} \leftarrow \textsc{PlanPaths}(\mathcal{G}, \mathcal{K})$ ▷ Candidate Trajectories
$\mathcal{Q} \leftarrow \emptyset$
$\mathcal{U} \leftarrow \mathcal{R}$ ▷ Uncovered targets
while $\mathcal{U} \neq \emptyset$ **do**
  $s^* \leftarrow$ null, $g_{max} \leftarrow 0$
  for $s \in \mathcal{S} \setminus \mathcal{Q}$ **do**
    $\Delta \leftarrow \textsc{Cover}(s) \cap \mathcal{U}$ ▷ Marginal coverage
    $g \leftarrow \textsc{Gain}(\Delta)$ ▷ Type-weighted gain
    if $g > g_{max}$ **then**
      $g_{max} \leftarrow g$
      $s^* \leftarrow s$
    end if
  end for
  if s^* is null **then**
    break
  end if
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{s^*\}$
  $\mathcal{U} \leftarrow \mathcal{U} \setminus \textsc{Cover}(s^*)$
end while
return $\textsc{ToTasks}(\mathcal{Q})$

Coverage Signature When executing a task, an agent necessarily visits a subset of rooms and interacts with a subset of entities; we treat these visited and exercised elements as the task's *coverage signature*. For example, the task "fetch an apple

from the kitchen" may require traversing the living room and bedroom and opening a chest, yielding a signature that covers {living room, bedroom, chest}. Formally, each candidate task $s \in \mathcal{S}$ is associated with a coverage signature $\textsc{Cover}(s) \subseteq \mathcal{R}$, which includes both the rooms traversed along the path and the interaction-relevant entities exercised by the task. Therefore, selecting a compact subset of tasks whose union covers $\mathcal{R}$ naturally induces a (weighted) set cover formulation.

Greedy Selection To improve data efficiency, we aim to minimize redundancy across tasks and keep the selected task set as compact as possible. We therefore adopt a greedy selection strategy that, at each step, chooses the task that contributes the largest marginal coverage over the yet-uncovered targets. A greedy weighted selection procedure is adopted in practice. At each iteration, the remaining uncovered set $\mathcal{U}$ is maintained, the marginal coverage is computed as $\Delta = \textsc{Cover}(s) \cap \mathcal{U}$. Then, the task maximizing a type-weighted gain $\textsc{Gain}(\Delta)$ is selected. The gain function assigns larger weights to **interactions** (w_i) and **objects** (w_o) than **rooms** (w_r) in a type-wise manner, prioritizing tasks that exercise complex behaviors over pure navigation. The procedure iterates until $\mathcal{U} = \emptyset$, yielding a minimal (approximate) task set $Q_{\min}$ with full target coverage and substantially reduced redundancy.

2.3 Quiz Generation

Beyond coverage-oriented tasks, a complementary quiz set is constructed to probe fine-grained grounded understanding of the environment. Question–answer pairs are generated from multiple perspectives.

Convert Environment Facts to Quiz Questions To automatically construct a comprehensive quiz suite that reflects an agent's understanding of and responsiveness to the environment, we generate quiz questions using deterministic rules grounded in the environment specification. Concretely, each environment instance is defined by a set of symbolic *facts* that encode fine-grained details such as entity locations, object attributes, room orientations, and connectivity (see Section A). We extract a fact list that covers all such environment details and convert each fact into one or more question–answer pairs. For example, the fact `in(apple, kitchen)` can be transformed into questions like "Where is the apple?" and "Is the apple in the kitchen?". This process yields a diverse quiz set that covers environment details across multiple question types.

Prerequisite Checkpoints In order to construct trajectory-conditioned, dynamic answers, prerequisite checkpoints are generated together with the questions.These checkpoints specify minimal interaction evidence required for an agent to be expected to know the answer. For example, a location question such as "the apple is in the kitchen" is only marked answerable if the agent has reached the kitchen.

Dynamic Ground Truth Generation Answers are assigned by combining static environment facts with the agent's interaction history.As Figure 3 shows, if all prerequisite checkpoints are satisfied, the agent is considered to have had sufficient opportunity to acquire the relevant knowledge, and the reference answer is used as the ground truth, which is regarded as "**should have known**" by the agent. Otherwise, the question is labeled as **non-answerable** for that trajectory to avoid penalizing agents for information they could not have observed. This hybrid of static answers and trajectory-conditioned answerability reached a balance between fairness and dynamic, behavior-aware evaluation.

3 Evaluation

3.1 Dataset

T2QBench We construct **T2QBench** via our pipeline and stratify environments into three difficulty levels based on the number of rooms and interactive objects. For each difficulty, we generate 10 distinct environments. Each environment is paired with a coverage-oriented task set and a quiz set. In total, the benchmark comprises 30 environments, 224 tasks, and 1,967 questions.

Quiz Taxonomy We categorize quiz questions into five types to probe complementary facets of grounded understanding:

- **Location (Loc.)**: asks where an entity is situated (e.g., the room/container of an object), which is often answerable given direct observation evidence.
- **Connectivity (Conn.)**: queries the existence of traversable links between locations under the environment topology.

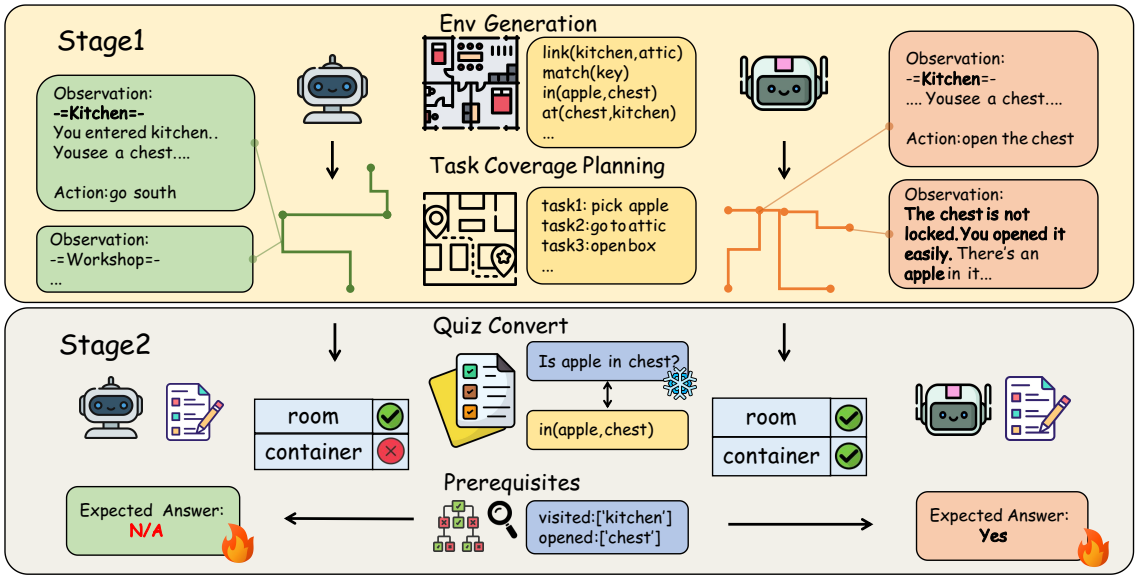

Figure 3: Pipeline of data construction and 2-stage evaluation. Data is generated in deterministic rules during multiple steps, such as room layout, object placement, task coverage planning, and quiz generation. Agent needs to complete a set of tasks first. Then, the agent's interaction history and the environment metadata are used to generate the quiz set. Finally, the agent is asked to answer the quiz set based on the their memory on stage 1.

- **Direction (Dir.)**: tests spatial orientation between locations typically demanding cross-turn aggregation and consistent coordinate reasoning.
- **Match (Match.)**: tests the agent's ability to determine which key is the correct one for the lock.
- **Property (Prop.)**: concerns latent object states, such as locked/unlocked, open/closed. Agent needs to interact with the object to reveal the property. For example, agent can only reveal a box is locked after trying to open it.

This taxonomy aligns generation with evaluation: each question is paired with trajectory-dependent prerequisites as described in Section 2.3 so that type-wise accuracy reflects reasoning/memory given sufficient exposure rather than missing exploration.

3.2 Formulation of Metrics

Stage 1: Task Success Rate In Stage 1, we evaluate an agent's performance on the generated task set, reporting TSR. Given an environment $\mathcal{E}$, we instantiate a task set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_{|\mathcal{T}|}\}$ by the coverage-oriented task generation logic as Algorithm 1. We then evaluate an agent (policy) π by executing each task τ_i in $\mathcal{E}$. The resulting interaction trace is a trajectory $\zeta_i = (s_0, a_0, s_1, a_1, \ldots, s_{T_i})$, and we denote the set of trajectories as $\mathcal{Z} = \{\zeta_1, \zeta_2, \ldots, \zeta_{|\mathcal{T}|}\}$. We define a task success indicator as $\mathbb{I}_{\text{win}}(\zeta_i, \tau_i) \in \{0, 1\}$, and report the task success rate:

$$\text{TSR} = \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \mathbb{I}_{\text{win}}(\zeta_i, \tau_i). \quad (1)$$

where $|\zeta_i|$ is the number of actions in ζ_i, and $\ell^*(\tau_i)$ is the oracle shortest action length to complete τ_i in $\mathcal{E}$ (when defined). For memory-augmented agents, Stage 1 interactions also update an internal memory state; we denote the resulting memory after executing the task set (associated with $\mathcal{Z}$) as $M_{\mathcal{Z}}$, which is carried into Stage 2.

Stage 2: Environment Understanding Score In Stage 2, we assess grounded understanding with quizzes spanning orientation, connectivity, object properties, and entity localization; importantly, a subset of questions depends on the agent's realized trajectory and thus requires interaction-conditioned reference answers. We further instantiate a question set $\mathcal{Q} = \{q_1, q_2, \ldots, q_{|\mathcal{Q}|}\}$ to probe the agent's grounded understanding of $\mathcal{E}$. Crucially, many questions are *trajectory-dependent*; therefore we construct reference answers by answerability analysis $g(\cdot)$ using both the question

and the realized interaction:

$$a_j^* = g(q_j, \mathcal{E}, \mathcal{Z}), \qquad \mathcal{A}^* = \{a_k^*\}_{k=1}^{|\mathcal{Q}|}. \quad (2)$$

Given $(q_j, M_{\mathcal{Z}})$, the agent produces a response with $\hat{a}_j = f_\pi(q_j, M_{\mathcal{Z}})$, yielding $\hat{\mathcal{A}} = \{\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_{|\mathcal{Q}|}\}$. We compute the Environment Understanding Score (EUS) as the ratio of the number of correct answers to the total number of questions:

$$\text{EUS} = \frac{1}{|\mathcal{Q}|} \sum_{j=1}^{|\mathcal{Q}|} \mathbb{I}\big(\hat{a}_j = a_j^*\big)\,. \quad (3)$$

Table 1: Dataset Statistics for T2QBench. For each difficulty level, we generate 10 environments with varying numbers of rooms and objects. Walkthrough is the number of steps in a golden solution to complete a task.

Setting	Easy	Medium	Hard
Rooms	3–5	6–10	16–20
Objects	6–10	14–18	28–32
Avg. Tasks	2.4	5.7	12.3
Avg. Walkthrough	4.17	5.25	5.78
Avg. Questions	34.6	65.6	94.7

4 Experiment

This section describes our experimental setup, including the evaluated models and agent baselines, followed by a discussion of the main results and associated analyses.

4.1 Models and Agent Baselines

We evaluate a strong proprietary model GPT-5.1 (OpenAI, 2025) and representative open-source models (DeepSeekV3.2 (DeepSeek-AI et al., 2025), ChatGLM4.6 (Team et al., 2025), and Qwen3-32B (Yang et al., 2025)). On top of each open -source backbone, we compare multiple agent configurations: a naive *In-context* baseline that retains the full interaction history, and memory-augmented baselines, including Mem0(Chhikara et al., 2025), LangMem(LangChain, 2024), and A-MEM(Xu et al., 2025) that implement different memory systems.

4.2 Main Result

We report both task completion performance and quiz accuracy on T2QBench across models and memory mechanisms. Quiz results are further broken down by question categories (e.g., direction, location, and object properties). Table 2 reveals consistent patterns across models, memory mechanisms, and question types. GPT-5.1 and DeepSeekV3.2 tend to achieve stronger task completion, whereas Qwen3-32B yields the best overall quiz-based environment understanding. In contrast, Mem0 and A-MEM often lag behind on both TSR and EUS, and the simple in-context baseline remains highly competitive. Looking into quiz categories, agents typically perform best on localization questions but struggle the most with object-property queries.

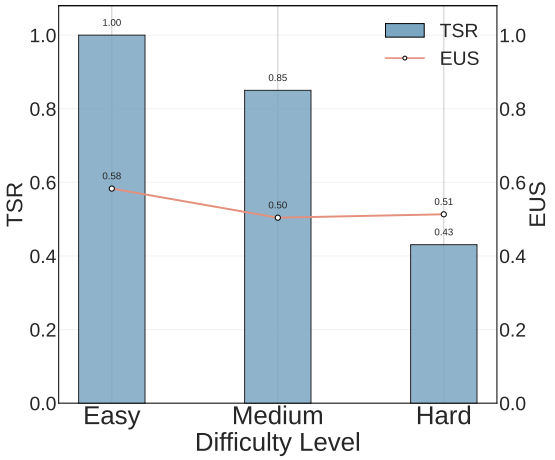

Figure 4: TSR and EUS trends across difficulty levels for **GLM-4.6** with the **in-context** method. TSR: task success rate; EUS: environment understanding score.

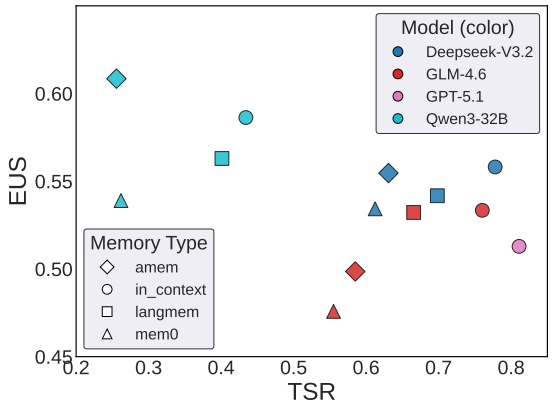

Figure 5: TSR and EUS Comparison: X-axis is the TSR score, Y-axis is the EUS score. The color represents the model, the shape represents the memory system.

4.3 Analysis

In this section, we analyze the main results and figure out the answers to the research questions.

Model	Method	Task Score	Environment Understanding Score					
			Loc.	Conn.	Dir.	Match.	Prop.	Tot.
GPT-5.1		**68.75**	62.52	51.66	45.68	43.53	35.93	50.23
GLM-4.6	In-context	61.16	62.85	56.40	53.33	38.24	35.65	52.41
	Mem0	39.29	55.32	52.84	50.37	24.71	28.41	46.21
	LangMem	51.34	60.23	59.72	57.04	33.53	30.08	51.65
	A-MEM	42.86	56.46	58.53	52.59	27.65	27.30	48.30
DeepseekV3.2	In-context	63.84	**68.41**	59.48	53.09	**49.41**	27.58	54.25
	Mem0	43.30	61.87	60.66	53.58	44.71	28.97	52.41
	LangMem	54.46	66.94	57.35	50.37	44.12	27.86	52.36
	A-MEM	45.54	66.94	59.00	56.05	46.47	30.36	54.55
Qwen3-32B	In-context	34.38	63.38	69.87	58.90	44.30	43.65	58.55
	Mem0	18.30	56.63	66.82	58.52	36.47	42.90	54.96
	LangMem	27.68	61.37	67.77	57.28	37.06	44.01	56.63
	A-MEM	18.30	61.05	**71.33**	**66.67**	44.12	**46.52**	**60.29**

Table 2: Main experiment results. Best and second-best values in each column are highlighted in **bold** and underlined, respectively, across all model–memory system combinations. Column abbreviations: Loc. (Location), Conn. (Connectivity), Dir. (Direction), Match (Matching), Prop. (Properties), Tot. (Overall).

RQ1: Different Trends of TSR and EUS Across Difficulty Levels To figure out the relationship between TSR and EUS, we evaluate the TSR and EUS trends across different difficulty levels. We observe that **TSR** and **EUS** exhibit markedly different behaviors as environment difficulty increases. Figure 4 illustrates this effect for GLM-4.6 under an in-context baseline: TSR saturates at 100% on easy environments but degrades substantially with increasing difficulty. In contrast, EUS remains comparatively stable across difficulty levels, suggesting that task success is not a reliable proxy for environment understanding.

Takeaway 4.1 for RQ1

Doing is different from knowing. Task success primarily tracks trajectory difficulty (e.g., longer solutions and tighter constraints), whereas EUS captures a more structural form of environment modeling grounded in interaction. Consequently, evaluating agents solely by completion metrics cannot fully reflect environment understanding.

RQ2: Comparison between In-context Method and Memory Systems We then examine the effect of memory systems on environment understanding. We compare a naive in-context baseline with several representative memory-augmented variants, and summarize TSR and EUS across models and memory systems in Figure 5. Surprisingly, the in-context approach often attains the strongest overall performance in both task success and environment understanding, outperforming methods equipped with explicit memory systems. This suggests that current memory systems do not reliably help agents form a more complete understanding of exploratory environments. A plausible explanation is that, during memory construction and retrieval, high-level abstractions or summaries discard critical fine-grained evidence, resulting in worse downstream reasoning than simply retaining the full interaction context.More broadly, limited environment mastery not only lowers EUS but also harms TSR, indicating that a comprehensive understanding of the environment is also important for tasks.

Takeaway 4.2 for RQ2

Current memory systems helps organization less than it hurts fidelity. Existing memory systems tend to lose environment-specific details while still failing to distill them into structural abstractions such as layout. This gap may stem from event-centric memory systems, highlighting the need for memory mechanisms tailored to grounded world-state representation.

RQ3: Comparison between Different Question Types To further investigate what hinders the agent's environment understanding, we analyze the performance of different question types in Figure 6. Models achieve relatively high accuracy on *location* questions, which are often directly answerable from a single recent observation. Performance drops for questions like orientation requir-

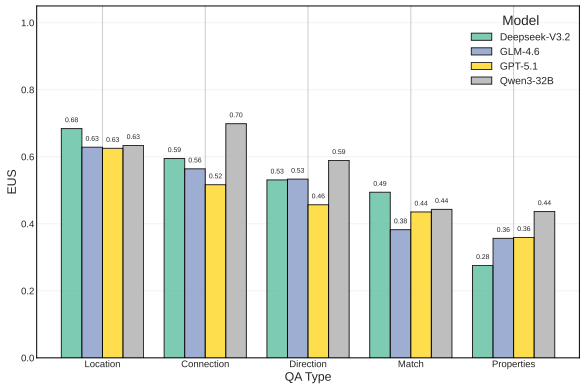

Figure 6: EUS Comparison Across Question Types: Agents have different performance on different question types.

ing multi-turn retrieval and reasoning across interactions. The largest degradation appears on questions that require *proactive* interaction to reveal latent properties or relations. Overall, these patterns suggest that insufficient exploration incentivizes is the dominant bottleneck than retrieval from memory.

Takeaway 4.3 for RQ3

Exploration is the dominant bottleneck than retrieval. Agents are typically optimized to complete a given task with minimal interactions, which incentivizes "efficient" goal-directed behavior. As a result, agents often fail to uncover environment details for deeper understanding. This suggests that the insufficient exploration incentives are the primary bottleneck than retrieval from memory.

5 Related Work

5.1 LLM Agentic Benchmarks

Agentic benchmarks increasingly evaluate LLM agents in *executable* environments that support multi-step interaction and tool use, such as realistic web navigation (Zhou et al., 2023; He et al., 2024) and repository-level software engineering tasks (Jimenez et al., 2023). More broadly, recent suites cover apps (Trivedi et al., 2024), OS/GUI-like scenarios (Agashe et al., 2024; Xie et al., 2024), and multi-environment or generalist evaluation (Liu et al., 2024; Mialon et al., 2023). Despite their achievements, many benchmarks remain task-centric and costly to scale, motivating more controllable and fine-grained evaluation beyond final success.

5.2 Game-based Evaluation

Games provide dynamic, multi-skill environments that stress long-horizon planning, exploration, and state tracking. Recent benchmarks include text/conversational games (Qiao et al., 2023; Costarelli et al., 2024), live computer games (Hu et al., 2024; Li et al., 2025), and multimodal games (Paglieri et al., 2024).Strategic and multi-agent settings further evaluate LLM decision-making (Wu et al., 2023; Duan et al., 2024; Chen et al., 2024) and broader gameplay ability (Huang et al., 2024; Chalamalasetti et al., 2023). Text-based adventure games are particularly attractive for grounded evaluation due to explicit state transitions and interpretable actions. Frameworks such as TextWorld (Côté et al., 2019) and interactive fiction resources (Hausknecht et al., 2020; Phan et al., 2025) become an alternative to mirror real-world challenges. In contrast to these evaluation works, we leverage text games as an framework to conduct the entire comprehensive evaluation of the agent's environment understanding.

6 Conclusion

In this paper , we introduces **Task2Quiz(T2Q)**, a new paradigm and automatically built **T2QBench**, a benchmark spanning three difficulty levels with 30 environments, 224 coverage-oriented tasks, and 1,967 environment-grounded QA pairs.We design a unified two-stage evaluation with Task Success Rate (TSR) and Environment Understanding Score (EUS).We present a comprehensive empirical study on **T2QBench**, and reveal **three key insights**: (i) There is a gap between 'knowing' and 'doing', task success rate can not reflect what agents *know* about the environment, as TSR degrades with difficulty while EUS remains comparatively stable; (ii) recent memory systems do not achieve better EUS than the naive in-context method, suggesting they are not yet effective at extracting and structuring environment-relevant information from interaction traces; and (iii) It's the lack of propensity for proactive exploration what hinders agents to form comprehensive world models. Beyond trajectory-level, our work offers a principled route to evaluate environment modeling capability of agents, providing a diagnostic foundation for developing more robust agents with generalized abilities.

Limitations

Despite our comprehensive analysis of agents' environment understanding, our work has several limitations.

First, due to computational constraints and the cost of API-based evaluation, we evaluate only a subset of representative open-source and closed-source models; extending coverage to a broader range of agents can improve the reliability of our findings.

Second, our benchmark is built on TextWorld framework (Côté et al., 2019), whose game mechanics and maps are relatively simple and purely text-based compared with commercial game such as Minecraft. Future extensions could incorporate more complex dynamics, larger maps, and additional modalities, such as visual or audio. There are two possible ways to do this, both challenging: (1) adapting a commercial game as the development environment, where access to copyright-protected source code is often restricted; and (2) building a new framework from scratch using a more powerful engine such as Unity (Unity Technologies, 2022) or Unreal Engine (Epic Games, 2023). The latter would require substantial engineering effort comparable to that of a game studio. Although our game framework is simple, it is sufficient to support the key insights of our work.

Finally, the world-analysis algorithm for coverage-oriented sub-tasks relies on TextWorld framework. It is implemented by a dependency tree structure and exhaustive state-space search (e.g., graph traversal over world states), which can become computationally expensive as the number of rooms and interactive objects increases for more complex games. Improving the efficiency and scalability of this algorithm remains a promising avenue for future research.

References

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2024. Agent s: An open agentic framework that uses computers like a human. *Preprint*, arXiv:2410.08164.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Kranti Chalamalasetti, Jana Götze, Sherzod Hakimov, Brielen Madureira, Philipp Sadler, and David Schlangen. 2023. clembench: Using game play to evaluate chat-optimized language models as conversational agents. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Junzhe Chen, Xuming Hu, Shuodi Liu, Shiyu Huang, Wei-Wei Tu, Zhaofeng He, and Lijie Wen. 2024. Llmarena: Assessing capabilities of large language models in dynamic multi-agent environments. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready AI agents with scalable long-term memory. *CoRR*, abs/2504.19413.

Anthony Costarelli, Mat Allen, Roman Hauksson, Grace Sodunke, Suhas Hariharan, Carlson Cheng, Wenjie Li, Joshua Clymer, and Arjun Yadav. 2024. Gamebench: Evaluating strategic reasoning abilities of llm agents. *arXiv preprint arXiv:2406.06613*.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, and 1 others. 2019. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers*. Springer.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A survey on code generation with llm-based agents. *Preprint*, arXiv:2508.00083.

Jinhao Duan, Renming Zhang, James Diffenderfer, Bhavya Kailkhura, Lichao Sun, Elias Stengel-Eskin, Mohit Bansal, Tianlong Chen, and Kaidi Xu. 2024. Gtbench: Uncovering the strategic reasoning capabilities of llms via game-theoretic evaluations. In *Advances in Neural Information Processing Systems*.

Epic Games. 2023. Unreal engine 5. `https://www.unrealengine.com`. Version 5.3, accessed 2025-01.

Matthew J. Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. 2020. Interactive fiction games: A colossal adventure. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI*

2020, New York, NY, USA, February 7-12, 2020, pages 7903–7910. AAAI Press.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*.

Lanxiang Hu, Qiyu Li, Anze Xie, Nan Jiang, Ion Stoica, Haojian Jin, and Hao Zhang. 2024. Gamearena: Evaluating llm reasoning through live computer games. *arXiv preprint arXiv:2412.06394*.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, Yuhuai Li, Shengze Xu, Shenzhi Wang, Xinchen Xu, Shuofei Qiao, Zhaokai Wang, Kun Kuang, Tieyong Zeng, Liang Wang, and 10 others. 2025. Os agents: A survey on mllm-based agents for general computing devices use. *Preprint*, arXiv:2508.04482.

Jen-tse Huang, Eric John Li, Man Ho Lam, Tian Liang, Wenxuan Wang, Youliang Yuan, Wenxiang Jiao, Xing Wang, Zhaopeng Tu, and Michael R Lyu. 2024. How far are we on the decision-making of llms? evaluating llms' gaming ability in multi-agent environments. *arXiv preprint arXiv:2403.11807*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

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

LangChain. 2024. Langmem: Long-term memory for llm agents. Accessed: 2025-12-28.

Zongyuan Li, Yanan Ni, Runnan Qi, Lumin Jiang, Chang Lu, Xiaojie Xu, Xiangbei Liu, Pengfei Li, Yunzheng Guo, Zhe Ma, Huanyu Li, Hui Wu, Xian Guo, Kuihua Huang, and Xuebo Zhang. 2025. Llm-pysc2: Starcraft ii learning environment for large language models. *Preprint*, arXiv:2411.05348.

Weiwen Liu, Jiarui Qin, Xu Huang, Xingshan Zeng, Yunjia Xi, Jianghao Lin, Chuhan Wu, Yasheng Wang, Lifeng Shang, Ruiming Tang, Defu Lian, Yong Yu, and Weinan Zhang. 2025. The real barrier to llm agent usability is agentic roi. *Preprint*, arXiv:2505.17767.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2024. Agentbench: Evaluating llms as agents. In *ICLR*.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, and 7 others. 2025. Large language model agent: A survey on methodology, applications and challenges. *Preprint*, arXiv:2503.21460.

Andrea Matarazzo and Riccardo Torlone. 2025. A survey on large language models with some insights on their capabilities and limitations. *Preprint*, arXiv:2501.04040.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2025. Large language models: A survey. *Preprint*, arXiv:2402.06196.

OpenAI. 2025. GPT-5.1. OpenAI API documentation. Accessed via OpenAI API.

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*.

Long Phan, Mantas Mazeika, Andy Zou, and Dan Hendrycks. 2025. Textquests: How good are llms at text-based video games? *Preprint*, arXiv:2507.23701.

Dan Qiao, Chenfei Wu, Yaobo Liang, Juntao Li, and Nan Duan. 2023. Gameeval: Evaluating llms on conversational games. *arXiv preprint arXiv:2308.10032*.

5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, and 152 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *Preprint*, arXiv:2508.06471.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*.

Unity Technologies. 2022. Unity real-time development platform. `https://unity.com`. Version 2022.3 LTS, accessed 2025-01.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. 2025. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. *Preprint*, arXiv:2505.16421.

Yue Wu, Xuan Tang, Tom Mitchell, and Yuanzhi Li. 2023. Smartplay: A benchmark for llms as intelligent agents. In *International Conference on Learning Representations*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-MEM: agentic memory for LLM agents. *CoRR*, abs/2502.12110.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. Survey on evaluation of llm-based agents. *Preprint*, arXiv:2503.16416.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *Preprint*, arXiv:2401.07339.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A The detail of Constructing an environment

We build each text-game environment with TextWorld's `GameMaker` API, which exposes the full world state by construction and allows us to precisely control layout, entities, and stateful relations. Concretely, an environment is specified through three steps: (i) defining the world topology (rooms and connectivity); (ii) instantiating entities (doors, containers, objects) and placing them; and (iii) declaring symbolic facts (e.g., locked/closed/match) that govern dynamics and answerable knowledge.

World topology (layout control). We explicitly create rooms and connect them via directional edges (e.g., `roomA.east` to `roomB.west`), yielding a controllable underlying graph. The player start location is also deterministically assigned. Since the topology is programmatically constructed, the ground-truth map (rooms and connectivity) is always available for verification and question generation.

Entities (object control). TextWorld provides typed entities that can be composed to form richer worlds. Rooms and fixed supports define where objects can appear, while doors and containers introduce stateful constraints (open/closed/locked) that shape exploration and interaction. Each entity can be named and optionally given a description used by `examine`.

Facts and relations (state control). Beyond entity placement, the environment dynamics and many evaluable properties are encoded as symbolic predicates added via `add_fact`. This makes the world fully controllable at the level of discrete state transitions (e.g., whether a door is locked) and relational structure (e.g., which key matches which lock), and provides a clean source of environment metadata for our verifier.

Supported entity types. We use the standard types provided by `GameMaker`:

- `r`: room; `d`: door; `c`: container; `s`: supporter
- `o`: portable object; `k`: key; `f`: food
- `oven`, `stove`: specialized heat-source container/supporter

Supported predicates (facts). We mainly rely on the following predicates for controllable state and relations:

- `match(key, door/container)`: key–lock compatibility
- `open(door/container)`, `closed(door/container)`, `locked(door/container)`: access state
- `edible(food)`: consumability (optional)

How to add new structure (examples). All controllable components are added deterministically in code, e.g., (i) create rooms and connect them; (ii) create entities and place them in a room/supporter/inventory; and (iii) attach facts to specify states and relations. For instance, a

locked door and its matching key can be created by adding `locked(door)` and `match(key, door)`, while containers can be initialized with `closed(container)` before gameplay.

Quests and distractors (task control). Given a constructed world, we can (a) record a specific quest by logging an intended command sequence, or (b) automatically sample multiple random quests of bounded length. We can also inject distractor objects to increase interaction complexity without changing the underlying ground-truth world state. Together, these controls allow us to scale environment instances while keeping their structure and metadata fully known and reproducible.

Minimal API examples. Below we provide short code snippets illustrating how we control topology, entities, and symbolic facts using TextWorld's `GameMaker`.

Example 1: Define rooms and connectivity (topology control)

```
M = GameMaker()
roomA = M.new_room("Room A")
roomB = M.new_room("Room B")
M.connect(roomA.east, roomB.west)
# directed ports define the map
M.set_player(roomA)
# fix the start room
```

Example 2: Add entities and place them (entity control)

```
s =M.new(type="s",name="table")
roomA.add(s)
# fixed in the room
```

Example 3: Add a locked door and matching key (fact/relation control)

```
d=M.new_door(corridor, name="door")
M.add_fact("locked", d)
k=M.new(type="k", name="old key")
M.add_fact("match", k, d)
s.add(k)
```

Example 4: Add a container with an explicit initial state

```
c=M.new(type="c", name="fridge")
M.add_fact("closed", c)
  # explicit initial status
roomB.add(c)
```

Entity types. We use TextWorld's typed object system, e.g., `r` (room), `d` (door), `c` (container), `s` (supporter), `o` (portable object), and `k` (key).

Controllable predicates. We encode state and relations as symbolic facts, including `open`, `closed`, `locked`, and `match(key, door/container)`. Because these facts are added by construction, the complete environment metadata (topology, placements, and states) is fully known and can be used for deterministic verification and question generation.

B Example of a game environment

Our generated game is packaged as a gym (Brockman et al., 2016) environment.Since all gym games have only one winning state.In order to apply our task sets on one environment, we actually generate a new game environment with the same structure but different goal. So one env $\mathcal{E}$ with n tasks $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ is actually organized as n games $\mathcal{G}_1 = (\mathcal{E}, t_1), \mathcal{G}_2 = (\mathcal{E}, t_2), \ldots, \mathcal{G}_n = (\mathcal{E}, t_n)$. The game content is shown in Figure 7. The task description is shown as follows:

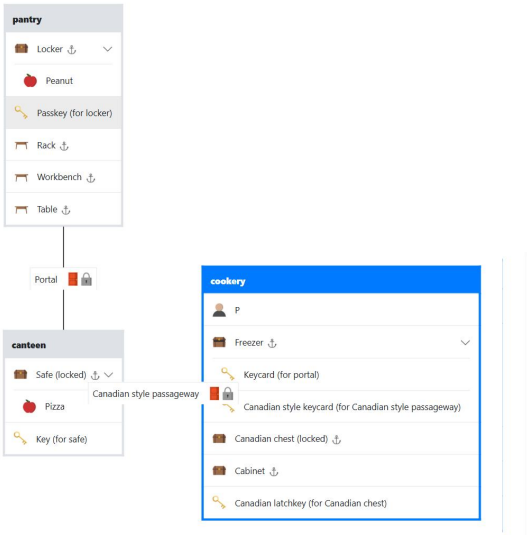

Figure 7: Example of a game environment

Task Description

```
Get ready to pick stuff up
and put it in places,
because you've just entered
TextWorld! I hope you're
ready to go into rooms and
interact with objects,
Recover the key from the
floor of the canteen.
```

C Prompt Templates

Our prompt templates are shown as follows:

QA Type-specific Guidelines

yes_no: Answer "yes" or "no" only.
which: Answer with the exact choice text from the given choices.
description: Provide a concise, categorical description. Focus on object types/categories rather than specific item names. Do NOT include exits, doors, or room connections. Valid options: [object, container, supporter, food, key].
where: Answer with the location/room name.
what: Answer concisely with the specific information requested.
default: Answer concisely.

Action Prompt Template

Observation:
{obs}

Inventory:
{inventory}

Score: {score}

Return ONLY JSON: {"reason": "...", "command": "<one command>"}

QA Prompt Template

You answer questions about TextWorld gameplay.
Based on the provided context, answer the following question about the TextWorld game initial state.

Context:
{context} (only included if context $\neq$ "")

{question}

Choices:
– {choice_1}
– {choice_2}
· · · (only included if choices is provided)

Answer should be json of the form:
{
"answer": "<your answer>",
"reason": "<why>"
}

Answer formatting guidelines:
{type_specific_guideline}

If the question cannot be answered based on the context, set answer to "non-answerable".

QA Type-specific Guidelines

yes_no: Answer "yes" or "no" only.
which: Answer with the exact choice text from the given choices.
description: Provide a concise, categorical description. Focus on object types/categories rather than specific item names. Do NOT include exits, doors, or room connections. Valid options: [object, container, supporter, food, key].
where: Answer with the location/room name.
what: Answer concisely with the specific information requested.
default: Answer concisely.

D The comparison of answerable QA accuracy and non-answerable QA accuracy

We compare the accuracy of answerable QA and non-answerable QA on the T2QBench dataset, as shown in Figure 8. In another word, we analyze the source of correctness. To figure out if agent can answer the exact fact ground truth, or gain accuracy simply by answering "I don't know".

From this figure, we observe that the composition of correctness sources varies substantially across models, particularly in the ratio of answerable vs. non-answerable QA. For location questions that can be resolved via single-turn retrieval, agents typically satisfy the prerequisite checkpoints during exploration and can answer correctly

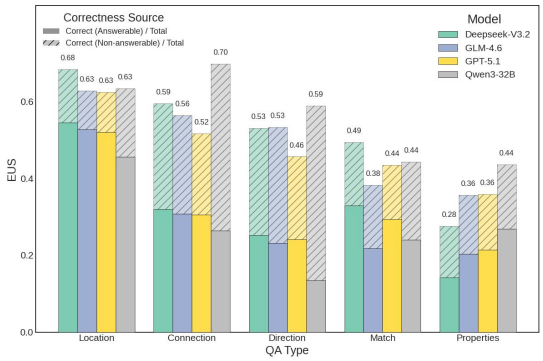

Figure 8: The comparison of answerable QA accuracy and non-answerable QA accuracy

once the relevant evidence is observed. In contrast, for orientation and connectivity questions, insufficient coverage of key locations often makes the required evidence unavailable; consequently, models can frequently obtain "correct" outcomes by predicting `non-answerable`. For object-property questions, however, models perform poorly on both answerable and non-answerable instances, suggesting that limited proactive interaction (needed to reveal latent properties) remains a major bottleneck.