MemPlan: Memory-Conditioned PDDL Planning for Partially Observable Text Environments

Tengwei Song[1,2,3,†] **Robert Hoehndorf**[1*]
[1] King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
[2] Åbo Akademi University, Finland
[3] ELLIS Institute Finland
songtengwei@gmail.com robert.hoehndorf@kaust.edu.sa

Abstract

LLM agents are flexible in language interaction, but using them as long-horizon planners is costly and prone to hallucinated or invalid actions. This paper targets a different role for LLMs: constrained bridges between language and symbolic planning. We propose MemPlan, a memory-conditioned PDDL planning framework for partially observable interactive text environments. MemPlan keeps the PDDL domain fixed and reconstructs the PDDL problem at each step from parsed observations, candidate hypotheses, and interaction memory. Positive graph memory assigns relevance-based action costs to candidate hypotheses for missing objects, locations, or conditions, guiding the planner toward more plausible checks. Negative memory records execution-refuted assumptions and compiles them into exclusion constraints, preventing repeated attempts at candidates already contradicted by feedback. A unified schema-conditioned language–symbol interface, trained offline with supervised fine-tuning, connects text environments to the planning loop through observation-to-fact parsing and symbolic-action-to-command grounding. Experiments on TextWorld, ALFWorld, and Robotouille show that MemPlan improves task completion and step efficiency over language-only and PDDL-aware baselines, while substantially reducing online token cost compared with LLM-planning baselines such as ReAct. The same fine-tuned interface is also reused across TextWorld task types without task-specific fine-tuning. Our code and data are available at: https://github.com/tewiSong/MemPlan.

1 Introduction

Interactive text environments require agents to solve long-horizon tasks under partial observability and delayed consequences, using natural-language observations and natural-language commands [Côté et al., 2018, Shridhar et al., 2021, Gonzalez-Pumariega et al., 2025]. Recent LLM agents can interpret textual feedback and generate plausible commands, but using LLMs as the primary planner remains costly and unreliable. In interactive settings, they may hallucinate actions, lose track of earlier observations, or fail to maintain a consistent long-horizon plan [Valmeekam et al., 2023, Sun et al., 2023, Zhang et al., 2025].

Classical planners offer a complementary strength. Given a PDDL domain, a symbolic state, and a goal, planners such as Fast Downward [Helmert, 2006] can search explicitly over action preconditions and effects [Aeronautiques et al., 1998]. Prior work empirically shows that replacing an LLM planner with a classical planner can save approximately 93.1% of LLM tokens [Dagan et al., 2023]. The difficulty is that partially observable text environments do not directly provide the complete

*Corresponding author.
†Work done while at KAUST.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

symbolic state required by a closed-world planner. Static LLM-to-PDDL formalization methods cannot interactively update the planning problem from environment feedback [Liu et al., 2023, Guan et al., 2023, Gestrin et al., 2024].

Existing work addresses this mismatch through two main routes. *Belief and open-world completion methods* make planning possible by first choosing or generating a completed symbolic state [Talamadupula et al., 2010, Dagan et al., 2023]. This approach works when the completion is reliable, but in text environments a missing object, location, or condition can have many plausible completions. Committing to one completed state before planning can lead to unnecessary search or repeated failed trials. *Iterative PDDL refinement methods* update domain or problem files from new observations and error feedback [Zhang et al., 2024, Gong et al., 2025]. This gives the representation more flexibility, but correctness then depends on repeated online formalization, which can introduce malformed PDDL, hallucinated facts, invalid goals, or forgotten observations.

This motivates a different use of symbolic planning under partial observability: a closed PDDL problem is still needed at each planner call, but the agent should avoid committing to a single guessed world state or relying on online LLM rewriting of PDDL files. To this end, we propose MemPlan, a memory-conditioned PDDL planning framework for partially observable text environments. MemPlan treats missing task-relevant state as candidate symbolic hypotheses and compiles interaction memory into the PDDL problem before each planner call. Positive graph memory ranks hypotheses for missing objects, locations, or conditions through action costs, allowing the planner to test more relevant candidates first. Negative memory records failed searches or refuted assumptions as symbolic exclusions, preventing repeated attempts at candidates already contradicted by the environment. A frozen schema-conditioned language–symbol interface connects text observations and commands to this symbolic loop through observation-to-fact parsing and symbolic-action-to-command grounding.

Our contributions are as follows. First, we introduce memory-conditioned PDDL problem construction, enabling a closed-world planner to operate in partially observable text environments without online PDDL rewriting. Second, we propose a positive/negative memory mechanism that ranks unresolved hypotheses with graph-derived costs and excludes execution-refuted candidates from later planning calls. Third, we train an offline schema-conditioned language–symbol interface for parsing observations and grounding symbolic actions, reducing online LLM use to constrained interface calls. We evaluate MemPlan on TextWorld, ALFWorld, and Robotouille, measuring task completion, step efficiency, token cost, and cross-task use of the learned interface.

2 Related Work

State, memory, and feedback in language agents. A large body of work studies how language agents act in interactive environments without an explicit symbolic planner. In text-based games, Keep CALM uses language models to generate action candidates and combines them with reinforcement learning for action selection [Yao et al., 2020]. More recent work focuses on maintaining internal state over long interactions. QuBE constructs question-based belief states to reduce reasoning derailment under partial observability [Kim et al., 2024], while WKM introduces a world-knowledge model with global task knowledge and local dynamic state knowledge for agent planning [Qiao et al., 2024]. AriGraph builds semantic and episodic graph memory for LLM agents in interactive text games [Anokhin et al., 2025]. Feedback-driven agents such as AdaPlanner refine language plans online using environment feedback [Sun et al., 2023]. These methods improve memory, action generation, or feedback use for LLM policies. MemPlan uses memory differently: graph-memory relevance is compiled into PDDL hypothesis-action costs, and execution-refuted assumptions are compiled into symbolic exclusions for a classical planner.

LLM-based PDDL formalization and planner interfaces. Another line of work connects language models with classical planning. LLM+P translates natural-language planning problems into PDDL, calls a classical planner, and translates the symbolic plan back into natural language [Liu et al., 2023]. Guan et al. use LLMs to construct PDDL world models and incorporate corrective feedback from validators and humans [Guan et al., 2023]. LLM-DP moves toward embodied interaction by using LLM-generated beliefs to sample plausible world states for symbolic planning [Dagan et al., 2023]. PDDLEGO and PDDLEGO+ study partial observability by iteratively formalizing, planning, growing, and refining PDDL representations through interaction [Zhang et al., 2024, Gong et al., 2025]. GenPlan studies a different interface: using pretrained LLMs to synthesize generalized

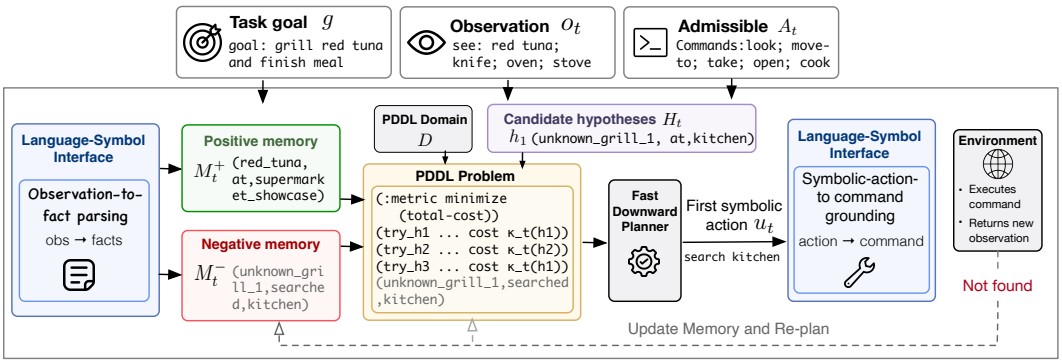

Figure 1: MemPlan online planning loop. Given the task goal, current observation, and admissible commands, the language–symbol interface extracts symbolic facts and grounds planner actions. MemPlan builds a PDDL problem from the fixed domain, candidate hypotheses, positive memory, and negative memory. Environment feedback updates memory and triggers replanning.

planners for PDDL domains [Silver et al., 2023]. These works show that LLMs can provide useful interfaces to PDDL planners, but they mainly focus on generating, completing, or refining planning representations. MemPlan assumes a fixed PDDL domain and studies how online interaction memory can be compiled into the PDDL problem through candidate hypotheses, action costs, and exclusions.

Open-world and structured embodied planning. Talamadupula et al. integrate a closed-world planner with an open-world robot using open-world quantified goals, runtime objects, execution monitoring, replanning, and soft goals [Talamadupula et al., 2010]. This work motivates the use of planner-internal placeholders for objects that may be discovered during execution. MemPlan adapts this idea to language-mediated environments, where both observations and executable actions are textual. The agent must therefore parse text observations into symbolic facts, ground symbolic actions back to admissible commands, and update memory from textual feedback. Recent structured embodied planning systems also use intermediate symbolic or graph representations: LLM-REGRESS performs open-world embodied planning with lifted regression and LLM-inferred affordances [Liu et al., 2025]; DELTA constructs planning problems from scene graphs and LLM decompositions [Liu et al., 2024]; LLM+MAP combines LLM reasoning with PDDL-style multi-agent planning for bimanual manipulation [Chu et al., 2025]; and HIVE uses a PDDL-empowered formal backbone for multimodal instruction execution [Vyas et al., 2025]. These systems highlight the value of structured representations, while MemPlan focuses on partially observable text environments and memory-conditioned PDDL problem construction.

3 Method

We study planning in partially observable text environments. At step t, the agent receives a task goal g, a textual observation o_t, and an admissible command set $\mathcal{A}_t$. The agent must select a command $c_t \in \mathcal{A}_t$ without access to the full environment state. MemPlan assumes a fixed PDDL domain $\mathcal{D}$, which specifies object types, predicates, action schemas, preconditions, and effects. The domain is provided to the agent and is not generated or repaired online.

Classical PDDL planners require a closed planning problem, while text environments expose only partial observations. MemPlan addresses this mismatch by rebuilding a closed PDDL problem at each step from parsed facts, candidate hypotheses, and interaction memory. Figure 1 summarizes the online loop. A frozen schema-conditioned language–symbol interface parses observations into symbolic facts, represented internally as typed triples, and grounds symbolic planner actions into executable commands. The planner searches over a PDDL problem constructed from the fixed domain, task goal, parsed facts, candidate hypotheses, positive memory, and negative memory.

Let F_t denote the validated triples extracted from o_t, $\mathcal{H}_t$ the candidate hypotheses for unresolved entities required by the goal, M_t^+ the positive memory, and M_t^- the negative memory. Let $\mathcal{P}_t$ denote the PDDL problem instance constructed for step t. MemPlan builds this problem as

$$\mathcal{P}_t = \text{BuildProblem}(\mathcal{D}, g, F_t, \mathcal{H}_t, M_t^+, M_t^-).$$

The planner returns a symbolic plan $\pi_t = (u_t^{(1)}, u_t^{(2)}, \ldots)$. MemPlan executes only the first symbolic action $u_t = u_t^{(1)}$, observes the resulting feedback, updates memory, and replans.

3.1 Offline Schema-Conditioned Language–Symbol Interface

MemPlan uses one language–symbol interface with two instruction formats. The interface is trained offline by supervised fine-tuning and frozen during evaluation. Online, it only performs observation-to-fact parsing and symbolic-action-to-command grounding. It does not generate plans, repair the PDDL domain, or rewrite PDDL problems.

Observation-to-fact parsing. Let $\mathcal{S}_\mathcal{D}$ denote the predicate and type schema induced by $\mathcal{D}$, and let $\mathcal{O}_t$ denote the current object glossary. Given $(o_t, \mathcal{S}_\mathcal{D}, \mathcal{O}_t)$, the parser outputs candidate triples. A triple is accepted into F_t only when its relation appears in $\mathcal{S}_\mathcal{D}$, its argument types match the schema, and its entities are grounded in $\mathcal{O}_t$ or introduced as placeholders used only inside planning. The parser extracts local symbolic evidence from the observation; MemPlan constructs the full PDDL problem.

Symbolic-action-to-command grounding. Given the first symbolic action u_t, the current observation o_t, and the admissible command set $\mathcal{A}_t$, the grounder scores each command $c \in \mathcal{A}_t$ with $s_\theta(u_t, o_t, c)$. Let $\mathcal{A}_t(u_t) \subseteq \mathcal{A}_t$ denote admissible commands that can realize u_t. When $\mathcal{A}_t(u_t)$ is nonempty, MemPlan selects

$$c_t = \arg \max_{c \in \mathcal{A}_t(u_t)} s_\theta(u_t, o_t, c).$$

When $\mathcal{A}_t(u_t)$ is empty, MemPlan selects an admissible observation command, updates memory only from environment feedback, and replans.

3.2 Memory and Candidate Hypotheses

Let $\mathcal{E}_t$ be the current entity set. MemPlan stores memory as typed triples over entities. Positive memory is $M_t^+ \subseteq \mathcal{E}_t \times \mathcal{R}_\mathcal{D} \times \mathcal{E}_t$, where $\mathcal{R}_\mathcal{D}$ is the set of positive relations defined in the PDDL domain. These triples are supported by observations or successful execution.

Negative memory is $M_t^- \subseteq \mathcal{E}_t \times \mathcal{R}_\mathcal{D}^- \times \mathcal{E}_t$, where $\mathcal{R}_\mathcal{D}^-$ contains relations that restrict future planning, such as `searched` and `not_at`. Negative triples are added after targeted checks whose feedback indicates that a candidate was searched and the required entity was not found.

When the goal requires an entity that has not been resolved, MemPlan introduces a typed placeholder z used only inside planning. A candidate hypothesis is a triple $h = (z, r, v)$, where r is a relation and v is a candidate entity, container, location, state, or condition. The candidate set $\mathcal{H}_t$ is generated from observed entities, memory graph neighborhoods, recently visited areas, and values compatible with the current goal. Once feedback resolves the placeholder to a concrete entity, later PDDL problems use the concrete entity.

This placeholder mechanism is inspired by open-world planning with runtime objects and execution monitoring [Talamadupula et al., 2010]. MemPlan adapts it to text environments by pairing placeholders with learned parsing, learned grounding, and memory-based hypothesis testing.

3.3 Memory-Conditioned PDDL Construction

BUILDPROBLEM converts the current interaction history into the closed PDDL problem $\mathcal{P}_t$. It writes grounded entities and unresolved placeholders into the object list; places unresolved placeholders in a virtual location such as `loc_limbo`; writes validated facts F_t and schema-compatible positive-memory triples M_t^+ into the initial state; inserts hypothesis-checking actions for candidates in $\mathcal{H}_t$; and compiles negative memory M_t^- into exclusions. The goal condition is derived from g, while all action schemas remain fixed in $\mathcal{D}$.

Positive memory as hypothesis costs. For each candidate $h = (z, r, v) \in \mathcal{H}_t$, MemPlan computes a relevance score $\mathrm{Score}_t(h) \in [0, 1]$ using frozen multilingual Contriever (mContriever) embeddings [Izacard et al., 2022]. The retrieval query is linearized from the task goal g and the unresolved

Algorithm 1 MemPlan online planning loop

Require: Fixed domain $\mathcal{D}$, goal g, initial observation o_0, frozen interface, planner PLAN

Initialize $M_0^+ \leftarrow \emptyset$, $M_0^- \leftarrow \emptyset$
for $t = 0, 1, 2, \ldots$ until termination **do**
  Receive o_t and admissible command set $\mathcal{A}_t$
  Parse and validate triples F_t from $(o_t, \mathcal{S}_\mathcal{D}, \mathcal{O}_t)$
  Add supported triples in F_t to M_t^+
  Generate candidate hypotheses $\mathcal{H}_t$ for unresolved entities required by g
  Compute $\mathrm{Score}_t(h)$ and $\kappa_t(h)$ for each $h \in \mathcal{H}_t$
  Remove candidates excluded by M_t^-
  Build $\mathcal{P}_t = \mathrm{BuildProblem}(\mathcal{D}, g, F_t, \mathcal{H}_t, M_t^+, M_t^-)$
  Compute $\pi_t \leftarrow \textsc{Plan}(\mathcal{D}, \mathcal{P}_t)$
  if π_t is empty **then**
    Choose an admissible observation command $c_t \in \mathcal{A}_t$
  else
    Let $u_t \leftarrow \pi_t[1]$
    Ground u_t to $c_t \in \mathcal{A}_t$; use an observation command if no admissible command realizes u_t
  end if
  Execute c_t and receive feedback and the next observation
  Update M_{t+1}^+ with supported triples and resolved placeholders
  Update M_{t+1}^- with searched or refuted triples from failed targeted checks
end for

placeholder z. The candidate document is linearized from the candidate triple h and the local neighborhood of v in the positive-memory graph induced by M_t^+. Here M_t^+ is a typed directed multigraph: observed or resolved entities are nodes, and each validated triple (u, r, v) creates a relation-labeled edge $u \xrightarrow{r} v$. Candidate documents use the one-hop neighborhood of v. The relevance score is derived from the cosine similarity between the query and candidate-document embeddings. We use the pretrained `facebook/mcontriever` checkpoint and apply min–max normalization to the cosine similarities over the candidate set. It is used only to rank hypothesis-checking actions. This use of dense graph-memory retrieval is related to AriGraph [Anokhin et al., 2025], but the retrieved relevance is used differently: MemPlan compiles it into PDDL action costs rather than using it as context for LLM decision-making.

MemPlan converts this score into a PDDL action cost. Physical actions, such as movement, pickup, and manipulation, receive the base cost κ_{base}. Hypothesis-checking actions test unresolved hypotheses before the planner commits to downstream physical actions. Their cost is

$$\kappa_t(h) = \kappa_{\text{base}} + \lceil \lambda \cdot \max(0, \tau - \mathrm{Score}_t(h)) \rceil ,$$

where $\kappa_t(h)$ is the cost of checking hypothesis h at step t, τ is a relevance threshold, and λ controls the penalty scale. Higher-relevance hypotheses receive lower costs. The generated PDDL problem includes `(:metric minimize (total-cost))`, so the planner is guided toward hypotheses better supported by positive memory. Since our planner configuration is satisficing, costs bias the returned plan without guaranteeing a globally minimum-cost plan. We fix $\kappa_{\text{base}} = 1$, $\tau = 0.50$, and $\lambda = 10$. A nine-setting sensitivity analysis is provided in Appendix A.7.

Negative memory as exclusions. When execution feedback refutes a candidate, such as a targeted search returning *not found*, MemPlan records the refutation in M_t^-. During the next call to BUILDPROBLEM, the refuted candidate is omitted or made inapplicable through exclusion predicates, such as `searched` or `not_at`. Thus M_t^+ controls which hypotheses are tried earlier, while M_t^- prevents the planner from repeating hypotheses already contradicted by the environment.

This construction differs from iterative PDDL rewriting methods. The PDDL domain remains fixed, and the LLM does not rewrite the domain or problem online. MemPlan rebuilds the problem by injecting placeholders, costing hypotheses with positive graph memory, and excluding refuted hypotheses with negative memory.

3.4 Online Planning and Memory Update

Algorithm 1 summarizes the online loop. MemPlan executes only the first grounded action of each plan, updates M_t^+ with supported observations and resolved placeholders, updates M_t^- with searched or refuted hypotheses, and replans after feedback.

4 Experiments

We evaluate MemPlan on interactive text environments with partial observability. The experiments address four questions. **Q1:** Does MemPlan improve task completion and step efficiency over language-only and PDDL-aware baselines, and how does its online LLM token cost compare with LLM-planning baselines? **Q2:** How does memory-conditioned PDDL planning compare with memory-augmented LLM agents? **Q3:** Can MemPlan handle synchronous and asynchronous long-horizon planning in Robotouille? **Q4:** Which components contribute to the observed gains?

4.1 Experimental Setup

Benchmarks. We evaluate MemPlan on TextWorld [Côté et al., 2018], ALFWorld [Shridhar et al., 2021], and Robotouille [Gonzalez-Pumariega et al., 2025]. ALFWorld is our main end-to-end benchmark; TextWorld is used for memory-agent comparison, cross-task interface use, and ablations; Robotouille tests synchronous and asynchronous long-horizon planning. We evaluate on 134 ALFWorld episodes, 7 TextWorld task instances, and 151 Robotouille task-seed runs. For Robotouille, success is first averaged over seeds within each task configuration and then macro-averaged across task configurations. Full split details, interaction budgets, metric definitions, and statistical reporting are provided in Appendix A.

PDDL domains. MemPlan uses one fixed PDDL domain per environment. We use the ALFWorld-released ALFRED domain and construct Robotouille domains from public domain metadata. TextWorld does not provide PDDL domains for the evaluated task families, so we generate its domains offline before evaluation. GPT-5.1 drafts each domain from environment-level task rules and action descriptions, *without* access to test-instance maps, object placements, oracle trajectories, interaction logs, or evaluation feedback. Validator and planner error messages are used for one offline repair round when needed. In our TextWorld protocol, $5/7$ domains (71.4%) are valid on the first attempt, and the remaining two are repaired successfully after one offline feedback round. The repaired domains are fixed for evaluation; MemPlan only rebuilds instance-level problem files online.

MemPlan implementation. Given the fixed domain for each environment, MemPlan constructs instance-level PDDL problems online from the current observation, task goal, positive memory, negative memory, and candidate hypotheses. Symbolic planning is performed by Fast Downward [Helmert, 2006]. The language–symbol interface uses Qwen2.5-7B as the backbone. We supervise fine-tuning one unified interface on TextWorld cook and hunt tasks, using 17,000 observation-to-fact examples and 9,600 symbolic-action-to-command examples, then freeze it for all evaluations. ALFWorld and Robotouille interface splits are used only for post-training diagnostic validation; they are not used for supervised fine-tuning, checkpoint selection, or hyperparameter tuning.

Baselines and metrics. We compare against language-only baselines, PDDL-aware baselines, and TextWorld memory-agent baselines. We report task success, environment steps, and online LLM token usage; metric definitions and evaluation protocols are detailed in Appendix A.

4.2 Q1: Main Results on ALFWorld

Table 1 examines performance across the six ALFWorld task types under the same 134-episode protocol, together with episode length and online LLM tokens. Per-task counts and statistical details are reported in Appendix A.3.

Table 1: ALFWorld task completion, episode length, and online LLM token usage.

Model	clean	cool	examine	heat	put	puttwo	overall ↑	length ↓	tokens ↓
LLM-DP	0.94	**1.00**	**1.00**	0.87	**1.00**	0.94	0.96	13.16	**633k**
GPT-4o	0.13	0.19	0.39	0.09	0.38	0.12	0.21	127.18	6.79M
GPT-4o + Domain	0.90	0.90	0.83	0.91	0.83	0.71	0.86	34.69	13.09M
ReAct (GPT-4o)	0.74	0.90	0.78	0.87	0.88	0.82	0.83	38.18	5.07M
ReAct (LLM-DP)	0.35	0.90	0.33	0.65	0.71	0.29	0.54	18.69	9.16M
MemPlan	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**6.91**	906k

Across the task types, giving GPT-4o domain descriptions improves completion but still leaves substantially longer trajectories than MemPlan. LLM-DP is more economical in online tokens, whereas MemPlan requires fewer environment steps. These differences illustrate the benefit of using the domain for explicit planning rather than only as prompt context.

To test whether the efficiency advantage persists against a stronger language-model agent and iterative PDDL refinement, Table 2 compares aggregate results on the same episodes. PDDLEGO+-Native retains its original fine-grained repair feedback; PDDLEGO+-Shared receives the standard textual feedback available to MemPlan.

Table 2: ALFWorld comparison with frontier LLM and PDDL-aware baselines (134 episodes).

Method	Success (%) ↑	Steps ↓	Tokens ↓
ReAct (GPT-5.1)	98.5	10.92	2.74M
PDDLEGO+-Native	94.8	14.32	2.31M
PDDLEGO+-Shared	92.5	15.47	1.82M
LLM-DP	96.0	13.16	**0.633M**
MemPlan	**100**	**6.91**	0.906M

Because frontier ReAct approaches ceiling-level completion on ALFWorld, the more informative differences are in interaction steps and online tokens. MemPlan produces shorter trajectories than the compared methods and uses fewer tokens than GPT-5.1 ReAct and both PDDLEGO+ variants. The PDDLEGO+ comparison remains favorable to MemPlan even under Native's more informative feedback; LLM-DP, however, retains the lowest online token cost.

4.3 Q2: TextWorld Memory-Agent Comparison

Figure 2 compares MemPlan with TextWorld memory-agent baselines from AriGraph [Anokhin et al., 2025]. These baselines use the same LLM planning and decision-making backbone but replace the memory module: FULL HISTORY keeps the complete observation–action history, SUMMARY maintains an iterative summary, RAG retrieves similar past memories, REFLEXION uses trial-level verbal reflection, SIMULACRA scores memories by recency, importance, and relevance, and ARIGRAPH uses a semantic–episodic memory graph. HUMAN ALL denotes average human performance.

MemPlan uses memory differently from these LLM-memory baselines. Positive memory shapes PDDL hypothesis costs, while negative memory excludes assumptions already contradicted by execution feedback. In the reported TextWorld comparison, MemPlan obtains the strongest normalized-score performance among the compared memory-agent methods in Figure 2. This directly answers Q2: memory-conditioned PDDL planning compares favorably with memory-augmented LLM agents on these TextWorld tasks. The result suggests that memory can be more effective when compiled into planning constraints and costs than when used only as additional LLM context.

Because this comparison uses 7 task instances, we interpret the result as evidence on the reported TextWorld setting rather than as a broad claim over all text-game distributions. Per-instance details and grouping information are reported in Appendix A.

4.4 Q3: Synchronous and Asynchronous Planning on Robotouille

Table 3 compares synchronous and asynchronous Robotouille tasks, with success macro-averaged over task configurations after averaging across seeds. MemPlan's lead over frontier ReAct and

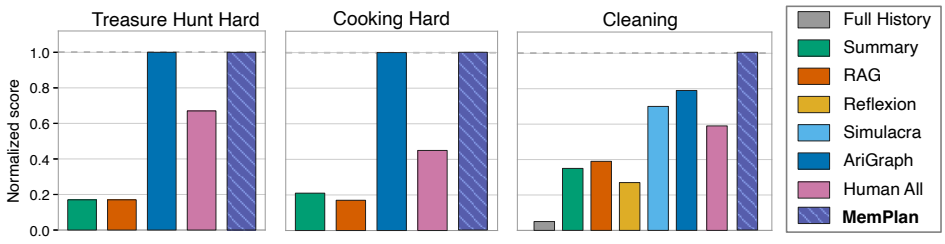

Figure 2: TextWorld normalized scores against memory-augmented LLM agents.

LLM-DP is more pronounced in the asynchronous setting. This is consistent with feedback-driven replanning being useful when action effects are delayed and previous observations become stale: MemPlan reconstructs the planning problem after each step, incorporates supported facts, and excludes refuted hypotheses.

Table 3: Robotouille task success in synchronous and asynchronous settings. Values are macro-averaged over task configurations after averaging over seeds within each configuration.

Model	Synchronous (%) ↑	Asynchronous (%) ↑
ReAct (GPT-4o)	47.0	11.0
ReAct (GPT-5.1)	63.6	24.7
LLM-DP	57.9	18.5
ReAct (GPT-4o-mini)	11.0	0.0
ReAct (Gemini-1.5-Flash)	0.0	0.0
ReAct (Claude-3-Haiku)	2.0	0.0
MemPlan	**71.2**	**41.3**

4.5 Q4: Ablation Study

Table 4 evaluates six ablations on the seven TextWorld task instances from Figure 2, with three seeds per instance. We report normalized score, executed steps, repeated hypothesis checks, terminal planner dead ends, and invalid actions. A terminal planner dead end means that the PDDL problem has no usable next symbolic action and the episode terminates. The ablations isolate graph-memory costs, negative memory, placeholders, observation parsing, action grounding, and the full learned interface. Protocol details are provided in Appendix D.

The ablations show three failure modes. Removing placeholders causes structural planning failure because required but unobserved entities cannot be represented in the PDDL problem. Removing negative memory mainly hurts recovery: the agent can still plan, but it revisits hypotheses already contradicted by feedback.

Interface ablations mainly affect execution reliability. Removing the parser weakens the symbolic state, while removing the grounder increases invalid commands. Removing both gives the weakest interface variant. Uniform-cost hypotheses preserve feasibility but require more interaction steps, indicating that graph-memory costs mainly improve search efficiency.

4.6 Efficiency Analysis

Table 5 reports ALFWorld efficiency in terms of environment steps, planner calls, LLM calls, online tokens, and invalid-action rate. Invalid-action rate is the fraction of executed commands rejected by the environment. MemPlan replans after each executed action, so planner calls follow episode length but do not consume LLM tokens. Its online LLM calls are limited to observation parsing and command grounding.

MemPlan uses more online tokens than LLM-DP, but achieves shorter episodes and higher task completion in Table 1. Detailed token-accounting rules are provided in Appendix A.8.

Table 4: TextWorld ablations. Values are means over 21 runs. Step count is omitted for terminal dead-end variants, where shorter length does not imply higher efficiency.

Model	Norm. score ↑	Steps ↓	Rep. checks / run ↓	Term. dead ends / run ↓	Invalid acts. / run ↓
MemPlan	**1.00**	**31.0**	**0.0**	**0.0**	**0.0**
Uniform-cost hypotheses	1.00	38.0	1.0	0.0	0.0
w/o negative memory	0.64	60.0	12.0	0.0	1.0
w/o placeholders	0.27	–	0.0	1.0	0.0
w/o observation parser	0.55	60.0	3.0	0.0	4.0
w/o action grounder	0.45	60.0	1.0	0.0	9.0
w/o learned interface	0.36	60.0	4.0	0.0	12.0

Table 5: ALFWorld online efficiency breakdown. Invalid-action rate is the fraction of executed commands rejected by the environment.

Model	Steps ↓	Planner calls ↓	LLM calls ↓	Tokens / episode ↓	Invalid action rate ↓
ReAct (GPT-4o)	38.18	–	39.78	37.9k	0.03
LLM-DP	13.16	13.16	13.16	4.7k	0.01
MemPlan	6.91	6.91	16.87	6.9k	0.00

4.7 Case Study: Recovering from a Missing Cooking Device

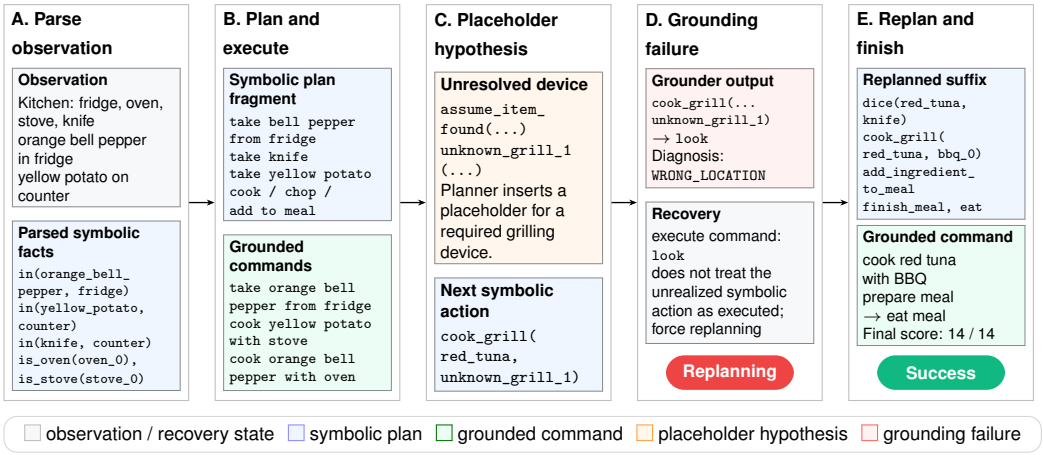

Figure 3: Case study on a TextWorld Cooking Hard task with an unresolved grilling device. MemPlan introduces a planner-side placeholder `unknown_grill_1`, uses feedback to avoid treating an unrealized symbolic action as executed, replans, and grounds the resolved device `bbq_0` to the executable command `cook red tuna with BBQ`.

Figure 3 shows a representative TextWorld Cooking Hard run completed by MemPlan in 31 steps with a normalized score of 1.00. The task requires collecting ingredients, applying preparation and cooking operations, assembling the meal, and eating it. MemPlan first parses textual observations into symbolic memory, then uses Fast Downward to produce a long-horizon PDDL plan.

The critical recovery occurs when the planner needs a grilling device that has not been grounded to a concrete object. MemPlan introduces the placeholder `unknown_grill_1` and plans a `cook_grill` action with it. Because the action cannot be grounded to an admissible command at the current step, MemPlan executes an observation command, does not treat the unrealized symbolic action as executed, and replans from updated memory. The replanned suffix resolves the device to `bbq_0`, whose executable environment command uses the surface argument `BBQ`. This allows the agent to execute `cook red tuna with BBQ` and finish the task.

5 Limitations

MemPlan has three main limitations. First, it assumes that a task-level PDDL domain is available and covers the relevant object types, predicates, action schemas, preconditions, and effects. Although TextWorld domains can be generated and repaired offline before evaluation accurately, MemPlan does not automate domain construction inside the online agent; incorrect or incomplete domain rules can still lead to invalid symbolic plans. Second, negative memory currently uses hard exclusions: once feedback marks a candidate hypothesis as refuted, the candidate is removed from later PDDL problems. This avoids repeated failed checks, but it can be brittle when feedback is ambiguous or incorrectly parsed, because a valid candidate may be excluded too early. Future work could replace this hard exclusion with confidence-weighted negative memory, where weak or ambiguous feedback lowers a candidate's priority instead of removing it immediately. Third, our evaluation is limited to simulated text environments with benchmark-specific action spaces and, when available, admissible commands. We do not evaluate safety-critical deployment, physical robotics, or open-ended environments without constrained command sets.

6 Conclusion

This work shows that a fixed-domain classical planner can be used effectively in partially observable text environments when interaction memory is made part of PDDL problem construction. MemPlan represents unresolved task-relevant information as candidate hypotheses, uses positive graph memory to assign planner-visible action costs, and converts execution-refuted assumptions into exclusions for later planning calls. This design keeps the LLM out of the online planning role and uses it only for constrained parsing and grounding. Across ALFWorld, TextWorld, and Robotouille, the results show that memory-conditioned PDDL construction improves task completion and step efficiency over language-only and PDDL-aware baselines. More broadly, MemPlan suggests a concrete role for memory in neural–symbolic agents: interaction memory can be compiled into planner-visible costs and constraints, rather than only retrieved as language context.

Acknowledgments and Disclosure of Funding

Funding. This work was supported by KAUST – Center of Excellence for Generative AI, under award number 5940. This work has also received funding from the European Union under the CEREBRA-AI project (Grant Agreement No. 101299046). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HaDEA). Neither the European Union nor HaDEA can be held responsible for them.

Competing interests. The authors declare no competing interests.

References

Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, et al. PDDL — the planning domain definition language, 1998.

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. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pages 12–20, 2025. URL `https://www.ijcai.org/proceedings/2025/0002`.

Kun Chu, Xufeng Zhao, Cornelius Weber, and Stefan Wermter. LLM+MAP: Bimanual robot task planning using large language models and planning domain definition language. *arXiv preprint arXiv:2503.17309*, 2025. URL `https://arxiv.org/abs/2503.17309`.

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. *arXiv preprint arXiv:1806.11532*, 2018. URL https://arxiv.org/abs/1806.11532.

Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a llm. *arXiv preprint arXiv:2308.06391*, 2023. URL https://arxiv.org/abs/2308.06391.

Elliot Gestrin, Marco Kuhlmann, and Jendrik Seipp. NL2Plan: Robust LLM-driven planning from minimal text descriptions. In *arXiv preprint arXiv:2405.04215*, 2024. URL https://arxiv.org/abs/2405.04215.

Liancheng Gong, Wang Zhu, Jesse Thomason, and Li Zhang. Iterative formalization and planning in partially observable environments. *arXiv preprint arXiv:2505.13126*, 2025. URL https://arxiv.org/abs/2505.13126.

Gonzalo Gonzalez-Pumariega, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille: An asynchronous planning benchmark for LLM agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OhUoTMxFIH.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=zDbsSscmuj.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006. URL https://www.jair.org/index.php/jair/article/view/10457.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022. URL https://openreview.net/forum?id=jKN1pXi7b0.

Minsoo Kim, Jongyoon Kim, Jihyuk Kim, and Seung-won Hwang. QuBE: Question-based belief enhancement for agentic LLM reasoning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21403–21423, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1193. URL https://aclanthology.org/2024.emnlp-main.1193/.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023. URL https://arxiv.org/abs/2304.11477.

Xiaotian Liu, Ali Pesaranghader, Hanze Li, et al. Open-world planning via lifted regression with LLM-inferred affordances for embodied agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025. doi: 10.18653/v1/2025.acl-long.1018. URL https://aclanthology.org/2025.acl-long.1018/.

Yuchen Liu, Luigi Palmieri, Sebastian Koch, Ilche Georgievski, and Marco Aiello. Delta: Decomposed efficient long-term robot task planning using large language models. *arXiv preprint arXiv:2404.03275*, 2024. URL https://arxiv.org/abs/2404.03275.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model. *arXiv preprint arXiv:2405.14205*, 2024. URL https://arxiv.org/abs/2405.14205.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2303.11366.

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

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. Generalized planning in pddl domains with pretrained large language models. *arXiv preprint arXiv:2305.11014*, 2023. URL https://arxiv.org/abs/2305.11014.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.16653.

Kartik Talamadupula, J Benton, Paul Schermerhorn, Subbarao Kambhampati, and Matthias Scheutz. Integrating a closed world planner with an open world robot: A case study. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 1561–1566, 2010.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - a critical investigation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=X6dEqXIsEW.

Kaustubh Vyas, Damien Graux, Yijun Yang, Sebastien Montella, Chenxin Diao, Wendi Zhou, Pavlos Vougiouklis, Ruofei Lai, Yang Ren, Keshuang Li, and Jeff Z. Pan. From an llm swarm to a pddl-empowered hive: Planning self-executed instructions in a multi-modal jungle. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=QAAsnSRwgu.

Shunyu Yao, Rohan Rao, Matthew J. Hausknecht, and Karthik Narasimhan. Keep calm and explore: Language models for action generation in text-based games. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8736–8754. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.704. URL https://aclanthology.org/2020.emnlp-main.704/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Li Zhang, Peter Jansen, Tianyi Zhang, Peter Clark, Chris Callison-Burch, and Niket Tandon. PDDLEGO: Iterative planning in textual environments. In *Proceedings of the 13th Joint Conference on Lexical and Computational Semantics (*SEM 2024)*, pages 212–221, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.starsem-1.17. URL https://aclanthology.org/2024.starsem-1.17/.

Wenxuan Zhang et al. MIRAGE-Bench: LLM agent is hallucinating and where to find them. *arXiv preprint arXiv:2507.21017*, 2025. URL https://arxiv.org/abs/2507.21017.

A Experimental Details

This appendix provides the experimental details for Section 4. It includes benchmark scope, episode budgets, metric definitions, statistical reporting, PDDL and planner settings, hypothesis-cost construction, and online token accounting.

A.1 Environment Roles and Evaluation Scope

We evaluate MemPlan on three interactive text environments. Each environment is used for a different experimental purpose.

ALFWorld. ALFWorld is the main end-to-end benchmark. It contains household manipulation tasks that require object search, receptacle reasoning, object-state changes, command grounding, and long-horizon execution. We evaluate on 134 ALFWorld episodes across six task types: clean, cool, examine, heat, put, and puttwo. Results are reported by task type and by macro-average over task types.

TextWorld. TextWorld is used to compare MemPlan against memory-augmented LLM agents and to evaluate cross-task use of the learned language–symbol interface. We evaluate on 7 TextWorld task instances, reported as three groups in Figure 2: Treasure Hunt Hard, Cooking Hard, and Cleaning.

Robotouille. Robotouille is used to evaluate synchronous and asynchronous long-horizon planning. We evaluate 151 task-seed runs. For each method and setting, success is first averaged over seeds within each task configuration and then macro-averaged across task configurations. Synchronous and asynchronous settings are reported separately. Each compared method uses the same task budget and termination rule for a given Robotouille task.

A.2 Episode Budgets and Evaluation Units

All compared methods use the same episode budget within each environment and task.

Table 6: Evaluation runs and episode budgets.

Environment	Evaluation unit	Episode budget
TextWorld	7 task instances	task-dependent limits
ALFWorld	134 episodes	150 steps
Robotouille	151 completed multi-seed runs	derived from each task's optimal-step budget

For ALFWorld, an episode is counted as successful if the environment reports task completion within 150 environment steps. For TextWorld, we follow the task-dependent limits and normalized-score protocol used by the compared memory-agent baselines. For Robotouille, a run is counted as successful if all required subtasks are completed within the task-specific budget.

Table 7: TextWorld evaluation grouping.

Group in Figure 2	Instance identifiers	Role in transfer evaluation
Treasure Hunt Hard	hunt, hunt_hard	fine-tuning task family
Cooking Hard	cook, cook_hard, cook_hardest, cook_rl_baseline	fine-tuning task family
Cleaning	clean_3x3	held-out task family

A.3 Metric Definitions and Statistical Reporting

Task success. Task success is 1 if the environment reports task completion within the episode budget and 0 otherwise. For ALFWorld, success is reported by task type and as a macro-average over the six task types. For Robotouille, success is reported separately for synchronous and asynchronous settings as a task-configuration macro-average after averaging over seeds.

Episode length. Episode length is the number of executed environment commands before success or budget exhaustion. Observation fallbacks, such as `look`, are counted as environment steps because they consume interaction budget.

Normalized score. For TextWorld, normalized score follows the protocol used by the compared memory-agent baselines. Scores are first computed for each task instance and then aggregated into the task groups reported in Figure 2.

Invalid actions. Invalid action rate is the fraction of executed commands rejected by the environment. For MemPlan, if a grounding output does not exactly match an admissible command, command normalization and admissible-command matching are applied before execution. If no admissible match exists and the command is executed and rejected by the environment, it is counted as invalid.

Repeated hypothesis checks. Repeated checks count targeted inspections or information-gathering actions that test a candidate already refuted earlier in the same episode. For example, checking the same unresolved object-location hypothesis after a prior "not found" feedback increments this metric.

Planner failures. Planner failures count cases where Fast Downward returns no plan, reaches the configured planning timeout, or returns a plan whose first symbolic action cannot be matched to any admissible command and cannot be replaced by an admissible observation command.

Online LLM tokens. Online LLM tokens count prompt and completion tokens used during test-time interaction. Fine-tuning tokens and offline data-construction tokens are excluded. Symbolic planner calls and deterministic PDDL construction are excluded because they do not invoke an LLM.

For success rates, we report point estimates and bootstrap confidence intervals over evaluation episodes or runs. For ALFWorld, bootstrap resampling is performed over the 134 episodes, with task-type macro-averages recomputed for each bootstrap sample. For Robotouille, bootstrap resampling is performed over completed multi-seed runs separately for synchronous and asynchronous settings. For TextWorld, because only 7 task instances are used, we report per-instance scores and group averages.

Unless otherwise stated, confidence intervals are two-sided 95% bootstrap intervals computed with 10,000 bootstrap resamples. When two methods are evaluated on matched episodes, we additionally report paired differences for success and episode length. These intervals quantify variation over the fixed benchmark split.

Table 8: Statistical reporting summary.

Environment	Resampling unit	Main metric	Main-text estimate	Appendix statistics
ALFWorld	episode	success, length, tokens	yes	95% CI, paired differences
TextWorld	task instance	normalized score	yes	per-instance scores, group averages
Robotouille	completed run	success	yes	95% CI by sync/async setting
Ablations	episode or run	success, length, diagnostics	yes	95% CI, paired differences

A.4 PDDL Domains and Planner

MemPlan separates fixed PDDL domains from online PDDL problem construction. The domain specifies object types, predicates, action schemas, preconditions, and effects. The problem file specifies the current objects, initial facts, goal facts, memory-conditioned candidate hypotheses, action costs, and exclusion constraints. During evaluation, MemPlan reconstructs only the problem file online; the domain file remains fixed.

The source of the fixed domain differs by benchmark. For ALFWorld, we use the benchmark-provided ALFRED PDDL domain released with ALFWorld. For Robotouille, we construct PDDL domains from public domain metadata and transition specifications. For TextWorld, because the benchmark does not provide PDDL domains for the evaluated task families, we generate domains offline before evaluation. The domain-generation prompt contains environment-level task rules and action descriptions only; it does not include test-instance maps, object placements, oracle trajectories, interaction logs, or evaluation feedback. GPT-5.1 drafts each domain, and validator or planner error messages are used for one offline repair round when needed. In our protocol, $5/7$ TextWorld domains

are valid on the first attempt, and the remaining two are repaired successfully after one feedback round. The resulting domains are fixed during evaluation and are not updated online.

MemPlan uses Fast Downward [Helmert, 2006] for symbolic planning. Generated problem files include:

```
(:metric minimize (total-cost))
```

The planner configuration is satisficing, so action costs bias hypothesis selection without guaranteeing globally optimal-cost plans. After each planner call, MemPlan executes only the first grounded command, observes textual feedback, updates memory, and replans when needed.

When an observation fallback is used, the unrealized symbolic action is not treated as executed. Memory is updated only from environment feedback.

A.5 Example Compiled PDDL Problem

The following listing shows one complete problem file generated by BUILDPROBLEM for the TextWorld Treasure Hunt run `pddl_output/hunt/job_46727745/problem_step_5.pddl`. The task is to obtain the target object `o_2`. This time step is useful because the problem contains planner-side placeholders, observation-derived memory facts, planner-visible candidate costs, and negative-memory exclusions from already searched rooms.

Negative memory is stored internally as typed triples over unresolved placeholders and candidate values. During PDDL problem construction, these triples are compiled into the predicates required by the fixed domain. In this Treasure Hunt domain, a refuted room candidate is represented by the unary predicate `searched(`r`)` because the exclusion only needs to block future checks of room r. Thus an internal record such as `(unknown_o_2, searched, r_0)` is compiled as `(searched r_0)` in the problem file.

```
(define (problem step_5)
  (:domain textworld)
  (:objects
    P - p
    I - i
    r_0 r_1 r_3 r_2 r_6 r_8 r_10 r_9 r_11 r_4 r_5 r_7
    unknown_r_1 loc_limbo - r      ; placeholder room + limbo for unknown locations
    c_0 c_1 c_10 c_11 c_2 c_3 c_4 c_5 c_6 c_7 c_8 c_9 - c
    k_0 k_1 k_2 k_3 o_2 unknown_k_1 - k
    o_0 o_1 unknown_o_1 unknown_o_2 - o
    unknown_f_1 unknown_f_2 - f
    unknown_d_1 - d
    unknown_t_1 unknown_t_2 - t
  )
  (:init
    (= (total-cost) 0)

    ; Positive memory: validated observation facts become ordinary init facts.
    (at P r_5)
    (at c_5 r_5)
    (at c_7 r_7)
    (at c_4 r_4)
    (at c_3 r_3)
    (at c_10 r_10)
    (at c_1 r_1)
    (at c_9 r_9)
    (at c_8 r_8)
    (at c_11 r_11)
    (at c_0 r_0)
    (at c_6 r_6)
    (at c_2 r_2)
    (locked c_2)
    (locked c_11)
    (locked c_5)
    (locked c_9)
    (locked c_0)
    (locked c_8)
    (locked c_7)
    (locked c_3)
    (locked c_1)
    (locked c_6)
    (locked c_10)
    (locked c_4)
```

```
(in o_2 c_0)
(in o_1 c_2)
(in k_3 c_9)
(in k_1 c_6)
(in o_0 c_6)
(in k_0 I)
(in k_2 c_2)
(match k_3 c_0)
(match k_2 c_9)
(match k_1 c_2)
(match k_0 c_6)
(match o_2 unknown_d_1)
(is_target_object o_2)
(is_player_inventory I)

; Placeholders for unresolved objects remain in limbo until assumed or grounded.
(at unknown_o_2 loc_limbo)
(at k_2 loc_limbo)
(at unknown_f_1 loc_limbo)
(at k_3 loc_limbo)
(at unknown_t_1 loc_limbo)
(at unknown_f_2 loc_limbo)
(at o_2 loc_limbo)
(at k_1 loc_limbo)
(at unknown_k_1 loc_limbo)
(at o_0 loc_limbo)
(at unknown_o_1 loc_limbo)
(at unknown_d_1 loc_limbo)
(at k_0 loc_limbo)
(at unknown_r_1 loc_limbo)
(at unknown_t_2 loc_limbo)
(at o_1 loc_limbo)
(is_unknown_area loc_limbo)
(is_frontier unknown_f_1)

; Positive memory supplies planner-visible candidate costs.
(= (hint_cost r_10) 1)
(= (hint_cost loc_limbo) 1)
(= (hint_cost r_9) 1)
(= (hint_cost r_8) 1)
(= (hint_cost r_2) 1)
(= (hint_cost r_5) 1)
(= (hint_cost r_11) 1)
(= (hint_cost r_3) 1)
(= (hint_cost unknown_r_1) 1)
(= (hint_cost r_7) 1)
(= (hint_cost r_6) 1)
(= (hint_cost r_4) 1)
(= (hint_cost r_1) 1)
(= (hint_cost r_0) 1)

; Negative memory: internal refutation triples are compiled into domain-level exclusion predicates.
; Here, searched(r) blocks repeated checks of a refuted room candidate.
(searched r_0)
(searched r_5)
(searched r_1)

; Navigation graph recovered from interaction memory.
(free r_3 r_6)
(west_of r_2 r_3)
(east_of r_8 r_10)
(free r_3 r_2)
(free r_7 r_6)
(free r_1 r_5)
(free r_6 r_11)
(free r_6 r_3)
(free r_9 r_8)
(free r_5 r_1)
(north_of r_1 r_2)
(free r_6 r_7)
(west_of r_1 r_0)
(free r_11 r_6)
(free r_2 r_3)
(east_of r_3 r_2)
(free r_2 r_4)
(free r_4 r_2)
(west_of r_3 r_6)
(free r_10 r_8)
(east_of r_6 r_3)
(free r_8 r_9)
(free r_7 r_8)
```

```
    (free r_8 r_7)
    (south_of r_11 r_6)
    (south_of r_4 r_2)
    (south_of r_2 r_1)
    (west_of r_10 r_8)
    (north_of r_5 r_1)
    (south_of r_7 r_8)
    (free r_8 r_10)
    (north_of r_7 r_6)
    (north_of r_6 r_11)
    (east_of r_9 r_8)
    (east_of r_0 r_1)
    (free r_1 r_0)
    (south_of r_1 r_5)
    (free r_1 r_2)
    (north_of r_8 r_7)
    (south_of r_6 r_7)
    (free r_0 r_1)
    (free r_2 r_1)
    (west_of r_8 r_9)
    (north_of r_2 r_4)
  )
  (:goal (and (in o_2 I)))
  (:metric minimize (total-cost))
)
```

The corresponding fixed domain action consumes the cost and exclusion facts supplied by the problem file:

```
(:action assume_item_found
  :parameters (?P - P ?o - o ?r - r ?unk - r)
  :precondition (and
    (at ?P ?r)
    (at ?o ?unk)
    (is_unknown_area ?unk)
    (not (= ?r ?unk))
    (not (searched ?r)))       ; compiled negative memory blocks refuted room candidates
  :effect (and
    (increase (total-cost) (hint_cost ?r)) ; graph cost biases hypotheses
    (not (at ?o ?unk))
    (at ?o ?r)
    (searched ?r)))     ; record that this room candidate has been checked
```

A.6 Graph-Memory Hypothesis Cost Construction

For each unresolved placeholder z, MemPlan constructs candidate hypotheses $h = (z, r, v)$ from entities and relations observed in the current episode. Candidate values are drawn from observed locations, containers, receptacles, objects, states, and task-related entities in positive memory. Candidates already refuted by negative memory are removed before PDDL problem construction. The internal negative-memory records are triples for consistency with hypothesis representation, but BUILDPROBLEM compiles them into the arity required by the target PDDL domain. For example, a placeholder-specific refutation may compile to `not_at(`z`,`v`)`, while a domain that only needs to block repeated room checks may compile the same evidence to a unary predicate such as `searched(`v`)`.

For each remaining candidate $h = (z, r, v)$, MemPlan computes a relevance score $\mathrm{Score}_t(h) \in [0, 1]$ as a text-based graph-memory retrieval score. The query string is

$$q_t(z) = \texttt{Goal: } \; g \; \texttt{; Missing: } \; z.$$

The candidate document is

$$d_t(h) = \texttt{Candidate: } \; (z, r, v) \; \texttt{; Context: } \; \mathrm{Linearize}(N_1(v; M_t^+)),$$

where $N_1(v; M_t^+)$ is the one-hop neighborhood of v in the positive-memory graph. The function $\mathrm{Linearize}(\cdot)$ serializes neighboring triples in a deterministic order. We encode $q_t(z)$ and $d_t(h)$ with the frozen mContriever encoder (checkpoint `facebook/mcontriever`) [Izacard et al., 2022] and compute cosine similarity:

$$\mathrm{sim}_t(h) = \cos\left(\mathrm{Enc}(q_t(z)), \mathrm{Enc}(d_t(h))\right).$$

The score is normalized over the current candidate set:

$$\mathrm{Score}_t(h) = \frac{\mathrm{sim}_t(h) - \min_{h' \in \mathcal{H}_t} \mathrm{sim}_t(h')}{\max_{h' \in \mathcal{H}_t} \mathrm{sim}_t(h') - \min_{h' \in \mathcal{H}_t} \mathrm{sim}_t(h') + \epsilon}.$$

We set $\epsilon = 10^{-8}$ for numerical stability. The score is used only to order hypothesis-checking actions. It does not generate symbolic plans, add memory facts, or decide whether a candidate is true.

MemPlan converts the score into a PDDL action cost:

$$\kappa_t(h) = \kappa_{\text{base}} + \lceil \lambda \cdot \max(0, \tau - \text{Score}_t(h)) \rceil ,$$

where κ_{base} is the base action cost, τ is the relevance threshold, and λ controls the penalty scale. Physical actions use κ_{base}. Hypothesis-checking actions receive larger costs when their relevance scores are lower than τ.

The generated PDDL problem includes:

```
(:metric minimize (total-cost))
```

Thus, the planner is biased toward checking hypotheses with stronger positive-memory support. Since the planner configuration is satisficing, these costs guide search without guaranteeing a globally minimum-cost plan. All cost parameters are fixed before test-time evaluation and are not tuned on test episodes. Negative memory can still remove a high-scoring candidate after it is refuted by execution feedback.

A.7 Positive-Memory Hyperparameter Sensitivity

We tested all nine combinations of $\tau \in \{0.40, 0.50, 0.60\}$ and $\lambda \in \{5, 10, 20\}$, with $\kappa_{\text{base}} = 1$. Across the evaluated TextWorld tasks, all nine settings achieved a normalized score of 1.00; mean episode lengths ranged from 31.0 to 34.1 steps. These results support the role of positive memory in ordering hypothesis checks and improving interaction efficiency. The ability to complete the tasks also depends on placeholders, negative memory, and the fixed PDDL domain.

A.8 Online Token Accounting

Online LLM tokens count prompt and completion tokens used during test-time interaction. For language-only baselines, this includes action-generation calls. For ReAct-style baselines, this includes reasoning and action-generation text. For LLM-DP, this includes online belief-state generation and action-selection text required during test-time interaction. For MemPlan, online LLM tokens come from Qwen2.5-7B interface calls for observation parsing and command grounding. Symbolic planner calls and deterministic PDDL construction are excluded because they do not invoke an LLM.

Offline costs are reported separately from online interaction costs. This includes supervised fine-tuning of the language–symbol interface and offline TextWorld domain generation/repair. The online token tables, therefore, measure test-time interaction cost, not one-time preprocessing cost.

Supervised fine-tuning cost is reported separately from online test-time token usage.

Table 9: Online token accounting. MemPlan uses language-model calls only for parsing and grounding.

Model	Action-generation tokens	Interface tokens	Planner tokens
GPT-4o	yes	no	no
GPT-4o + Domain	yes	no	no
ReAct	yes	no	no
LLM-DP	yes	no	no
MemPlan	no	yes	no

Table 10: Per-episode online efficiency statistics.

Model	Steps / episode	LLM calls / episode	Tokens / episode	95% CI for tokens / episode
ReAct (GPT-4o)	38.18	39.78	37.9k	[34.8k, 41.2k]
LLM-DP	13.16	13.16	4.7k	[4.2k, 5.3k]
MemPlan	6.91	16.87	6.9k	[6.4k, 7.5k]

B Interface Training and Validation Details

B.1 Unified Interface Model

MemPlan uses Qwen2.5-7B fine-tuned as a unified schema-conditioned language–symbol interface. The same frozen model handles two task formats: observation-to-fact parsing and symbolic-action-to-command grounding.

Observation-to-fact parsing. The input contains a textual observation, the domain predicate schema, and an object glossary. The output is a set of PDDL-compatible facts. Outputs are checked against valid predicate names and typed object identifiers before they are used to update memory or construct the next PDDL problem.

Symbolic-action-to-command grounding. The input contains a symbolic planner action, the current textual observation, and the admissible command set. The output is an executable text command. This turns command generation into candidate selection from admissible commands.

B.2 Training Data

The Qwen2.5-7B interface is fine-tuned on 17,000 observation–fact examples and 9,600 symbolic-action–command examples. Observation–fact labels come from simulator annotations. Symbolic-action–command labels are constructed through heuristic alignment between planner actions, oracle trajectories, and admissible commands.

Table 11: Training data for the unified Qwen2.5-7B language–symbol interface.

Training task format	Number of examples	Label source
Observation-to-fact parsing	17,000	simulator annotations
Symbolic-action-to-command grounding	9,600	planner-action / oracle-trajectory / admissible-command alignment

B.3 TextWorld Interface Transfer Setting

For TextWorld, the Qwen2.5-7B interface is fine-tuned only on cook and hunt task families. After fine-tuning, the interface is frozen and used inside the full MemPlan loop for the remaining TextWorld task types. No examples from target task types are used for supervised fine-tuning.

Table 12: TextWorld interface-transfer setting. Cooking Hard and Treasure Hunt Hard correspond to the fine-tuning task families, while Cleaning evaluates cross-task use of the frozen interface.

Fine-tuning task families	Evaluation task groups in Figure 2	Interface update during evaluation
cook, hunt	Cooking Hard, Treasure Hunt Hard, Cleaning	none

B.4 Interface Validation Splits

The interface is trained only on TextWorld cook and hunt task families. The ALFWorld and Robotouille validation splits reported in Table 13 are used only for post-training diagnostic evaluation of parsing and grounding behavior. They are not used for supervised fine-tuning, checkpoint selection, prompt tuning, threshold tuning, or any test-time adaptation. The same frozen checkpoint is used in all end-to-end evaluations.

B.5 Output Validation

For observation parsing, predicted facts are accepted only if they use valid predicates and typed object identifiers from the current domain schema and object glossary. Invalid facts are discarded before memory update.

For command grounding, the model selects an admissible command. If the output does not exactly match an admissible command, we apply command normalization and admissible-command matching. Commands rejected by the environment are counted in the invalid-action metric.

B.6 Interface Validation Metrics

We evaluate the interface separately from the end-to-end planning loop. The interface is trained only on TextWorld cook and hunt task families; ALFWorld and Robotouille validation examples are used only for post-training diagnostic evaluation and are not used for supervised fine-tuning, checkpoint selection, prompt tuning, threshold tuning, or test-time adaptation. Observation parsing is evaluated using precision, recall, and F1 over accepted PDDL-compatible facts. Command grounding is evaluated using top-1 admissible-command accuracy. We also report schema rejection and environment rejection rates to measure invalid parsed facts and commands rejected by the environment.

Table 13: Validation metrics for the schema-conditioned language–symbol interface.

Environment / split	Parsing precision	Parsing recall	Parsing F1	Grounding top-1	Schema rejection	Env. rejection
ALFWorld validation	0.94	0.92	0.93	0.96	0.03	0.01
TextWorld validation	0.96	0.94	0.95	0.97	0.02	0.01
Robotouille validation	0.91	0.89	0.90	0.94	0.04	0.02

B.7 Interface Error Categories

We classify interface errors into the following categories.

Parser omission. A true observation-supported fact is missing from the parser output.

Parser hallucination. The parser outputs a fact that is not supported by the observation. Schema validation removes malformed or type-invalid hallucinations before they can update memory.

Entity mismatch. The parser or grounder selects the wrong object identifier, receptacle, or location among valid candidates.

Predicate mismatch. The parser uses a domain-valid predicate whose relation type does not match the observation.

Grounding mismatch. The grounder selects an admissible command whose surface form is executable but does not correspond to the intended symbolic action.

No safe grounding. The symbolic action contains an unresolved placeholder or context-dependent object that cannot be grounded to an admissible command. In this case, MemPlan executes an admissible observation command when available. The unrealized symbolic action is not treated as executed, and memory is updated only from environment feedback.

Table 14: Interface error breakdown.

Split	Parser omission	Parser hallucination	Entity mismatch	Predicate mismatch	Grounding mismatch	No safe grounding
ALFWorld	28	14	18	10	20	10
TextWorld	24	12	16	9	28	11
Robotouille	30	16	20	12	14	8

B.8 Training Hyperparameters

Table 15 reports interface training hyperparameters.

C Baseline Details

C.1 Baseline Configurations

We compare against language-only baselines, including GPT-4o, GPT-4o with domain descriptions, and ReAct [Yao et al., 2023]; PDDL-aware baselines, including LLM-DP [Dagan et al., 2023]; and

Table 15: Training hyperparameters for the unified Qwen2.5-7B interface model.

Item	Value
Base model	Qwen2.5-7B-Instruct
Fine-tuning method	LoRA, rank 16, alpha 32, dropout 0.05
Training epochs	2
Learning rate	5×10^{-5}
Batch size	4 per device, gradient accumulation 4
Maximum input length	131,072 tokens tokenizer limit
Maximum output length	no separate cap; target shares the 131,072-token sequence limit
Optimizer	AdamW, linear schedule, no warmup, weight decay 0
Checkpoint selection	final checkpoint after 2 epochs; intermediate checkpoints saved every 166 steps

TextWorld memory-agent baselines from AriGraph [Anokhin et al., 2025]. Baseline prompts, model versions, admissible-command access, domain-information access, history windows, and stopping rules are fixed before evaluation.

C.2 PDDLEGO and PDDLEGO+

PDDLEGO and PDDLEGO+ are the closest prior methods that iteratively construct or refine PDDL representations during interaction. The published PDDLEGO+ results use a different trial set and manually implemented fine-grained simulation errors for online PDDL repair. We evaluate PDDLEGO+ on the 134 ALFWorld episodes in two configurations (Table 2). PDDLEGO+-Native retains the original fine-grained repair feedback. PDDLEGO+-Shared receives the same standard textual environment feedback available to MemPlan, without the additional error channel. Apart from feedback access, both use the task goals, episode budget, admissible-command access, and stopping criteria in our evaluation protocol. Both retain iterative PDDL formalization, planning, execution, and refinement. We account for all online PDDL generation and repair calls in token usage.

C.3 Common Evaluation Protocol

All baselines are evaluated with the same task goals, textual observations, episode limits, and admissible-command access when the benchmark exposes admissible commands. Baselines differ in how they use language models, memory, domain information, and symbolic planning.

The stopping rule is shared across methods: the episode terminates when the environment reports success, the method reaches the episode budget, or the benchmark-specific failure condition is triggered. Prompt templates, model versions, decoding parameters, history-window sizes, and domain-description inputs are fixed before evaluation and are not tuned on test episodes.

Table 16: Information available to each method at test time. "Benchmark-dependent" means that admissible-command access is used only when the corresponding benchmark exposes it and is applied consistently across compared methods within that benchmark.

Model	Goal / observation	Interaction history	Admissible commands	Domain information	Symbolic planner
GPT-4o	yes	yes	benchmark-dependent	no	no
GPT-4o + Domain	yes	yes	benchmark-dependent	natural-language domain description	no
ReAct	yes	recent history	benchmark-dependent	no	no
LLM-DP	yes	yes	benchmark-dependent	PDDL-aware belief planning	yes
PDDLEGO / PDDLEGO+	yes	yes	benchmark-dependent	iterative PDDL construction/refinement	yes
MemPlan	yes	symbolic memory	yes when exposed	fixed PDDL domain and predicate schema	yes

These comparisons should be interpreted as system-level comparisons among language-only, PDDL-aware, and memory-conditioned PDDL-planning agents. MemPlan assumes a fixed PDDL domain and studies how interaction memory can be compiled into online PDDL problem construction.

C.4 Language-Only Baselines

GPT-4o. GPT-4o receives the task goal, current observation, and interaction history. It outputs one text command per step. When admissible commands are provided by the benchmark, they are

included in the prompt using the same admissible-command access policy used by other language-only baselines.

GPT-4o + Domain. GPT-4o + Domain receives the same input as GPT-4o, plus a natural-language description of task-relevant action types, object types, and environment rules. It does not receive a machine-readable PDDL problem file or planner output.

ReAct. ReAct [Yao et al., 2023] interleaves reasoning and action generation. The model receives the task goal, current observation, and recent interaction history, then outputs one command to execute. The reasoning text is counted in online LLM tokens. ReAct (GPT-5.1) follows the same ALFWorld and Robotouille evaluation protocols, including the same task configurations and seeds where applicable (Tables 2 and 3).

C.5 PDDL-Aware Baselines

LLM-DP. LLM-DP [Dagan et al., 2023] uses LLM-generated belief states for symbolic planning. We evaluate it with the same episode limits and environment observations used for MemPlan. Its online LLM tokens include belief-state generation and action-selection text required during test-time interaction.

PDDLEGO and PDDLEGO+. PDDLEGO [Zhang et al., 2024] and PDDLEGO+ [Gong et al., 2025] iteratively construct or refine PDDL representations in partially observable textual environments. We report the same-protocol PDDLEGO+-Native and PDDLEGO+-Shared comparisons in Table 2; they differ in whether the original fine-grained repair errors or standard environment feedback is used.

C.6 TextWorld Memory Baselines

For TextWorld, we compare MemPlan with memory-augmented LLM agents when published protocols are compatible. These baselines include full-history context, summary memory, retrieval memory, Reflexion [Shinn et al., 2023], Simulacra-style simulation memory, and AriGraph [Anokhin et al., 2025]. These baselines use memory to support LLM decision-making. MemPlan uses memory differently: it compiles graph-memory relevance into PDDL hypothesis-action costs and compiles execution-refuted candidates into exclusions for later problem construction.

C.7 Prompt and Decoding Configuration

Table 17 summarizes the prompt and decoding configuration used for LLM baselines and for the MemPlan interface. The exact prompt templates are included in the released code.

Table 17: Prompt and decoding configuration.

Model / component	Model version	Temperature	Max output tokens	History window
GPT-4o	gpt-4o	0.0	512	full episode history
GPT-4o + Domain	gpt-4o	0.0	512	full episode history + domain description
ReAct (GPT-4o)	gpt-4o	0.0	512	full episode history
ReAct (LLM-DP)	gpt-4o	0.0	512	belief-state history
MemPlan parser	Qwen2.5-7B fine-tuned	0.0	1024	current observation
MemPlan grounder	Qwen2.5-7B fine-tuned	0.0	1024	current observation + admissible commands

D Ablation Details

Ablations are evaluated on the same seven TextWorld task instances used in Figure 2: `hunt`, `hunt_hard`, `cook`, `cook_hard`, `cook_hardest`, `cook_rl_baseline`, and `clean_3x3`. Each variant is run with three seeds per instance, yielding 21 runs per variant and 147 runs in total. All variants use the same PDDL domains, planner settings, candidate-generation rules, and interaction budgets unless the ablated component changes them.

D.1 Ablation Variables

Uniform-cost hypotheses. Replaces graph-memory hypothesis costs with equal costs. Candidate generation, placeholders, negative memory, parsing, grounding, and replanning are unchanged.

w/o negative memory. Removes searched and refuted triples from PDDL problem construction, allowing previously contradicted hypotheses to be checked again.

w/o placeholders. Constructs PDDL problems only with observed concrete objects. Required but unobserved entities cannot be represented as planner-internal placeholders.

w/o observation parser. Removes the fine-tuned observation-to-fact parser while keeping the learned action grounder and all memory/planning components fixed.

w/o action grounder. Removes the fine-tuned symbolic-action-to-command grounder while keeping the learned observation parser and all memory/planning components fixed.

w/o learned interface. Removes both fine-tuned interface components.

D.2 Ablation Metrics

Normalized score. TextWorld scores are normalized by the maximum achievable score of each task instance and averaged over 21 runs per variant.

Steps. Number of executed environment commands. Runs that reach the interaction budget without completion are assigned the budgeted step count.

Repeated checks. Mean number of targeted checks per run that revisit hypotheses already refuted earlier in the same run.

Planner dead ends. Mean number of cases per run where the constructed PDDL problem yields no usable next symbolic action.

Invalid actions. Mean number of executed commands per run that are rejected by the environment.

E Case-Study Grounding Details

The case study in Figure 3 contains both planner-side symbols and environment-side command arguments. The placeholder `unknown_grill_1` denotes an unresolved planner-side grilling device. After replanning, the unresolved device is resolved to a concrete planner-side object `bbq_0`. The executable environment command uses the surface argument `BBQ`, yielding the command `cook red tuna with BBQ`.

This distinction matters because the internal planner object name and the text environment's command argument need not be string-identical. The grounding module maps the symbolic action and its arguments to one command from the admissible command set.

F Additional Reproducibility Details

Compute resources. All MemPlan experiments were run on Intel Xeon Platinum 8260 CPU nodes with NVIDIA Tesla V100-SXM2 GPUs. Each V100 has 32 GB GPU memory. The completed Qwen2.5-7B interface fine-tuning job used 8 V100 GPUs, 8 CPU cores, and 32 GB allocated system memory for 62.4 hours, corresponding to approximately 500 GPU-hours. The evaluation jobs used V100-class nodes with 120–160 GB allocated system memory. End-to-end evaluation over the reported ALFWorld, TextWorld, and Robotouille runs took approximately 21 wall-clock hours across the final evaluation campaigns. Fast Downward planning was run on CPU, while language–symbol interface calls used the fine-tuned Qwen2.5-7B model. The reported compute excludes preliminary experiments that did not affect the final reported results.

Code and data availability. We release code, configuration files, PDDL construction scripts, evaluation scripts, and reproduction instructions at `https://github.com/tewiSong/MemPlan`. The experiments use existing benchmark environments: TextWorld, ALFWorld, and Robotouille. The released code documents environment setup, dependency installation, model-interface usage, and commands for reproducing the main tables.

Licenses and existing assets. We use existing benchmark environments, models, and software tools, including TextWorld, ALFWorld, Robotouille, Fast Downward, Qwen2.5-7B, GPT-4o, ReAct, and LLM-DP. The released MemPlan code is distributed under the MIT License. TextWorld is released under the MIT License. ALFWorld is released under the MIT License. Robotouille is released under the MIT License. Fast Downward is released under the GNU General Public License v3.0. Qwen2.5-7B-Instruct is released under the Apache License 2.0. We did not vendor external ReAct, LLM-DP, PDDLEGO, or PDDLEGO+ source code in this release; the corresponding baselines are either locally implemented from the published method descriptions or reported only where published protocols are compatible. We cite the corresponding papers in the main text and follow the licenses and terms of use of the original assets.

LLM usage. LLMs are used as components of the method and baselines. In MemPlan, the fine-tuned Qwen2.5-7B interface is used for observation-to-fact parsing and symbolic-action-to-command grounding. GPT-4o and ReAct-style baselines use LLMs for online action generation. LLM-DP uses LLM-generated belief states for symbolic planning. No LLM is used to generate or repair MemPlan's PDDL domain during evaluation.

Broader impacts. This work studies memory-conditioned symbolic planning for language agents in simulated text environments. A potential positive impact is improved reliability and token efficiency for long-horizon language agents by limiting LLM use to constrained language–symbol interfaces and using explicit symbolic planning. Potential negative impacts include over-reliance on generated symbolic states, failures caused by incorrect parsing or grounding, and misuse of autonomous planning systems in settings where incorrect actions may have real-world consequences. We mitigate these risks by evaluating only in simulated environments, using explicit PDDL constraints, maintaining execution-refuted negative memory, and analyzing failure modes. The method is not evaluated for safety-critical deployment, physical robotics deployment, or decision-making about people.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the main contributions: memory-conditioned PDDL planning, positive graph memory for hypothesis costs, negative memory for exclusion constraints, and schema-conditioned language–symbol interfaces. The experimental section evaluates these claims on TextWorld, ALFWorld, and Robotouille in Section 4.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix F discusses limitations and risks, including failures under partial observability, incorrect parsing or grounding, over-reliance on generated symbolic states, and the fact that the method is not evaluated for safety-critical deployment.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present theoretical results requiring formal assumptions or proofs. The equations in Section 3 define the planning framework, interface modules, and memory update procedure.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4 describes the environments, evaluation protocol, baselines, metrics, planner, and interface model. Appendices A, B, C, and F provide additional evaluation, training, baseline, compute, and reproduction details, and the anonymized code and data are available at `https://github.com/tewiSong/MemPlan`.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The paper provides an anonymized public code and data link at `https://github.com/tewiSong/MemPlan`. Appendix F states that the released materials include configuration files, PDDL construction scripts, evaluation scripts, and reproduction instructions.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 specifies the main experimental setup, including environments, episode counts, baselines, metrics, interface model, and planner. Appendices A–C provide additional evaluation, data, transfer, baseline, output-validation, and training details, including the interface-training hyperparameters in Table 15.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Appendix A.3 reports the statistical procedure used for the main experimental metrics. We report two-sided 95% bootstrap confidence intervals with 10,000 resamples for ALFWorld and Robotouille, use episode or completed-run resampling units as appropriate, and additionally report paired differences when methods are evaluated on matched episodes; for TextWorld, we report per-instance scores and group averages because only 7 task instances are used.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix F reports the compute resources used for training and evaluation, including CPU type, GPU type, GPU memory, allocated system memory, fine-tuning GPU-hours, evaluation wall-clock time, and the CPU/GPU division between Fast Downward planning and Qwen2.5-7B interface calls.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research evaluates planning agents in simulated text environments and uses existing benchmark environments. The paper does not involve human-subject experiments, private data collection, or deployment on real users.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix F discusses positive impacts, including improved reliability and token efficiency for long-horizon language agents, and negative impacts, including possible failures under partial observability and misuse of autonomous planning systems.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper releases code and data for a domain-specific planning framework evaluated in simulated text environments. It does not release a high-risk pretrained language model, image generator, scraped dataset, or other asset requiring special misuse safeguards.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the original works for the main existing assets, including TextWorld, ALFWorld, Robotouille, Fast Downward, Qwen2.5-7B, GPT-4o, ReAct, and LLM-DP. Appendix F states the license information for the used assets, including MIT licenses for TextWorld, ALFWorld, and Robotouille, GPLv3 for Fast Downward, Apache License 2.0 for Qwen2.5-7B-Instruct, and MIT License for the released MemPlan code.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces and releases the MemPlan code and data as new assets at `https://github.com/tewiSong/MemPlan`. Appendix F states that the released repository includes configuration files, PDDL construction scripts, evaluation scripts, reproduction instructions, and an MIT License for the released MemPlan code.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing experiments or research with human subjects. The experiments are conducted in simulated benchmark environments.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human-subject research, so IRB approval or equivalent review is not applicable.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The paper explicitly uses LLMs as core method components. Section 3.1 describes the offline-trained language–symbol interface for observation-to-fact parsing and symbolic-action-to-command grounding, Section 4.1 states that Qwen2.5-7B is fine-tuned and frozen for evaluation, and Appendix B provides the interface training and validation details.