Fact-Augmented Lookahead Planning for LLM Agents

Samuel Holt*
University of Cambridge

Max Ruiz Luyten*
University of Cambridge

Thomas Pouplin
University of Cambridge

Mihaela van der Schaar
University of Cambridge

Abstract

Large Language Models (LLMs) are increasingly capable, but LLM agents still struggle to plan effectively in interactive, partially observable, long-horizon environments when search is unguided or recent history is insufficient. We introduce LWM-Planner, a fact-augmented lookahead planning framework that improves agent behavior purely through in-context learning. After each episode, the agent extracts task-critical atomic facts from its trajectories, validates candidates with a lightweight predictive-consistency filter (and optionally compresses them), and uses the resulting fact set to condition action proposal, single-step latent world-model simulation, and state-value estimation. Planning then proceeds via recursive, depth-limited lookahead over candidate trajectories conditioned on the accumulated facts and recent history, enabling online improvement without parameter updates. We provide abstraction-style motivation—treating facts as reducing state aliasing (proxy ϵ_{sim}) and fact-conditioned simulation as lowering one-step error (proxy δ_{model})—without claiming formal guarantees. Empirically, on text FrozenLake variants, CrafterMini, and ALFWorld, the approach improves cumulative return over ReAct/Reflexion and search-only baselines, suggesting that additional test-time search is most useful when grounded by compact, experience-derived facts.

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated remarkable potential in building autonomous agents for sequential decision-making in diverse settings, from text-based games (Yao et al., 2023) to complex interactive environments (Yang et al., 2025). A key insight underpinning their success is their vast pre-trained knowledge, which can be steered towards specific tasks. However, effectively harnessing this knowledge and enabling LLMs to learn from new experiences in-context remains a critical challenge for improving their accuracy and optimality in long-horizon tasks.

Many LLM agents rely on extensive few-shot examples (Shinn et al., 2023) or retrieve entire past trajectories (Kagaya et al., 2024) to inform their decisions. While effective to a degree, these approaches can lead to very long prompts or may not efficiently distill the most crucial pieces of information from past experiences. Model-based approaches (Hao et al., 2023; Chae et al., 2024) have emerged, but often involve learning separate, potentially shallow, predictive models or require environment interactions for each step of their lookahead, rather than leveraging the LLM's inherent simulation capabilities more deeply.

The core idea of this paper is that LLMs possess a substantial amount of latent knowledge about world dynamics and task structures. To unlock better planning, we need to identify and provide the missing pieces of information—concise, critical insights derived from experience—that allow the LLM to more accurately simulate outcomes and evaluate states. We propose a novel LLM agent architecture that learns and utilizes "atomic facts" to augment its planning process. These facts are textual statements (e.g., "object X is in receptacle_Y", "action Z leads_to_failure_condition") extracted from the agent's interaction history at the end of each episode.

Our agent employs these atomic facts to inform a re-

*Equal contribution.

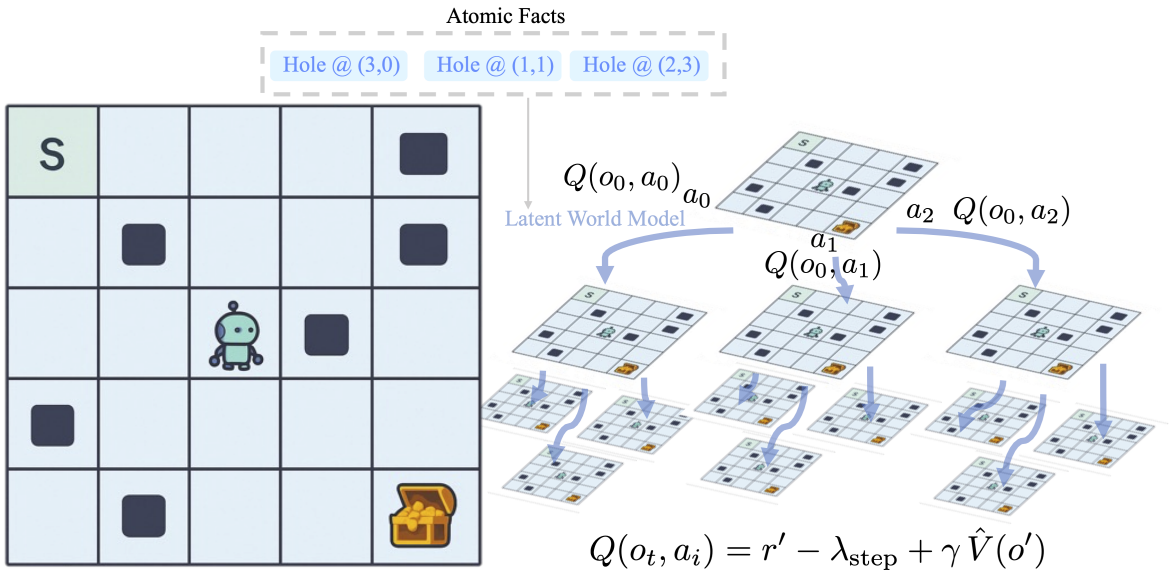

Figure 1: **LWM-Planner extracts compact facts from experience and uses them to ground lookahead planning at test time.** Starting from the current environment observation o_0 (left panel), the agent conditions on previously extracted `Atomic Facts` (e.g., `Hole@(3,0)`) and performs a recursive lookahead search (right panel) to choose its next action. The search combines: (i) an LLM, acting as a *Latent World Model*, to simulate action sequences (a_i) and predict subsequent latent states (o') and immediate rewards (r'); (ii) an LLM-based state-value estimator that provides $\hat{V}(o')$ at the search frontier; and (iii) Q-value aggregation from the simulated outcomes and value estimates via $Q(o_t, a_i) = r' - \lambda_{\text{step}} + \gamma \hat{V}(o')$, which guides action selection from o_0.

cursive, depth-limited lookahead search. The planning process involves three key LLM-driven components: An **action proposer** to suggest plausible next actions. A **latent world model** to simulate the next observation, reward, and termination status given an action. A **value estimator** to predict the long-term utility of states, especially at the leaves of the search tree. All these components receive the current observation, recent interaction history, and the curated set of atomic facts as input, allowing the LLM to make more informed predictions and decisions. The agent learns online, purely in-context, as the set of facts evolves with experience, leading to improved policies without any LLM fine-tuning. This approach is inspired by Dyna-style architectures (model-learning + planning) (Sutton, 1990), where experience is used to refine a model (here, the fact-augmented LLM reasoning process) which is then used for planning.

Contributions: ①**Algorithmic:** We present a simple fact-augmented planning mechanism for LLM agents: after each episode the agent distills a small set of atomic, task-specific facts and conditions *action proposal*, *single-step simulation*, and *value estimation* within a depth-limited lookahead (Section 3). A lightweight *predictive-consistency* filter and optional compression keep the fact set minimal and within the context window. ② **Theoretical Framing:** We connect the design to fact-based state abstraction (Section 2): facts aim to reduce state aliasing (a proxy for ϵ_{sim}) while fact-conditioned simulation targets lower one-step prediction error (a proxy for δ_{model}), which together can reduce planning sub-optimality ϵ_{plan}. These links are provided as motivation rather than formal guarantees. ③ **Empirical:** Across text environments (TextFrozenLake, CrafterMini) and the ALFWorld evaluation suite, we compare against strong search and agent baselines (ReAct, Reflexion, Tree-of-Thought, RAP), reporting cumulative return. Targeted ablations indicate that grounding lookahead with learned facts is the dominant contributor to the gains, and we analyze computation–performance trade-offs.

Taken together, these results suggest that in partially observable, multi-step environments, compact facts learned from experience can make test-time lookahead materially more reliable.

Figure 1 gives a high-level overview of the resulting planning loop.

This work offers a step towards LLM agents that can more effectively learn from their interactions in-context, leading to more robust and accurate planning by systematically augmenting the LLM's reasoning with distilled, experience-grounded knowledge.

2 THEORETICAL FRAMEWORK FOR FACT-BASED REINFORCEMENT LEARNING

We propose enabling agents to construct and reason over a *fact-based world model*. Such a model relies on a compressed, symbolic representation of task-relevant information extracted as "atomic facts" from the environment's state. We do so only as design motivation: we define an ideal fact-based agent and derive bounds that motivate Section 3; these are not guarantees for LWM-Planner, and we do not estimate or optimize ϵ_{sim}, δ_{model}, or ϵ_{plan}.

2.1 Problem Formulation

We model the agent's interaction with its environment as a **Markov Decision Process (MDP)**, specified by the tuple $\mathcal{G} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$. Here, $\mathcal{S}$ is the set of environment states, which we assume are fully observable or derivable into a sufficient structured representation $s_t \in \mathcal{S}$ from raw observations o_t. $\mathcal{A}$ is a finite set of actions. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability function, $\mathcal{T}(s'|s, a) = P(s_{t+1} = s'|s_t = s, a_t = a)$. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, with expected reward $R(s, a) = \mathbb{E}_{s' \sim \mathcal{T}(\cdot|s,a)}[R(s, a, s')]$. Finally, $\gamma \in [0, 1)$ is the discount factor. The agent's goal is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected discounted cumulative reward, $V^\pi_{\mathcal{G}}(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k})|s_t = s\right]$. The optimal value function in $\mathcal{G}$ is $V^*_{\mathcal{G}}(s) = \max_\pi V^\pi_{\mathcal{G}}(s)$.

To manage the potential complexity of $\mathcal{S}$, we introduce a **fact-based state abstraction**.

Definition 2.1 (Fact-Based Abstraction)**.** *Let $\mathcal{F}_{atomic}$ be a vocabulary of atomic predicates (e.g., `is_goal(loc)`, `obstacle_at((x,y))`) relevant to the task. These predicates primarily describe properties of the current state s_t. A **fact set** for a state $s \in \mathcal{S}$ is $F_s = \{f \in \mathcal{F}_{atomic} \mid f \text{ is true in } s\}$. The **Fact Extractor** is an abstraction function $\Psi : \mathcal{S} \to \mathcal{Z}_{\mathcal{F}}$, where $\mathcal{Z}_{\mathcal{F}} = \mathcal{P}(\mathcal{F}_{atomic})$ is the space of all possible fact sets. Each $z \in \mathcal{Z}_{\mathcal{F}}$ constitutes an **abstract state**, representing the equivalence class of ground states $\{s \in \mathcal{S} \mid \Psi(s) = z\}$. We denote the abstract state at time t as $z_t = \Psi(s_t)$. A critical objective is that $|\mathcal{Z}_{\mathcal{F}}| \ll |\mathcal{S}|$.*

This abstraction Ψ induces an **abstract MDP** $M_\Psi = (\mathcal{Z}_{\mathcal{F}}, \mathcal{A}, \mathcal{T}_\Psi, R_\Psi, \gamma)$, where $\mathcal{T}_\Psi(z'|z, a)$ and $R_\Psi(z, a)$ are the abstract transition and reward functions. These are derived from $\mathcal{G}$ by averaging over the ground states s that map to a given abstract state z (Li et al., 2006). For instance, if $\mathcal{S}_z = \{s' \in \mathcal{S} \mid \Psi(s') = z\}$, then $R_\Psi(z, a) = \frac{1}{|\mathcal{S}_z|} \sum_{s \in \mathcal{S}_z} R(s, a)$, assuming a uniform distribution over ground states within an abstract state.

2.2 An Idealized Fact-Based Agent (IFBA)

We conceptualize an Idealized Fact-Based Agent (IFBA) that flawlessly leverages such abstractions.

Definition 2.2 (Ideal Fact Abstraction Ψ^*)**.** *The IFBA employs an ideal abstraction function $\Psi^* : \mathcal{S} \to \mathcal{Z}_{\mathcal{F}}$ which establishes an ϵ_{sim}-**approximate bisimulation** with the ground MDP $\mathcal{G}$ (Ferns et al., 2004). This implies a bound on the difference between the optimal value functions in $\mathcal{G}$ and the induced abstract MDP M_{Ψ^*}:*

$$\|V^*_{\mathcal{G}} - V^*_{M_{\Psi^*}} \circ \Psi^*\|_\infty \leq \frac{\epsilon_{sim}}{1 - \gamma} \tag{1}$$

where $V^*_{M_{\Psi^*}}$ is the optimal value function for M_{Ψ^*}, and $\epsilon_{sim} \geq 0$ quantifies the maximum one-step deviation in rewards and discounted next-state distributions for states aggregated by Ψ^*.

As we will discuss later, we would like it to be **minimal**, achieving the sufficiency for near-optimal value representation (Eq. (1)) with the smallest possible fact set. This aligns with the Information Bottleneck (IB) principle (Tishby et al., 2000; Alemi et al., 2017), which seeks a compressed representation $z_t = \Psi^*(s_t)$ of an input s_t that maximizes information about a target variable (e.g., $V^*(s_t)$ or future returns) while minimizing $I(z_t; s_t)$ (or a proxy like fact set complexity).

The IFBA is assumed to dynamically maintain such an abstraction.

Definition 2.3 (Ideal Abstract World Model and Planner for IFBA)**.** *The IFBA is endowed with:*

1. ***A perfect abstract world model:** Its internal model $\hat{M}_{\Psi^*}$ is identical to the true abstract MDP M_{Ψ^*}, implying zero model error w.r.t. M_{Ψ^*}.*
2. ***An ϵ_{plan}-optimal planner:** This planner computes a policy $\pi^\circ_{M_{\Psi^*}}$ for M_{Ψ^*} such that $V^*_{M_{\Psi^*}}(z) - V^{\pi^\circ_{M_{\Psi^*}}}_{M_{\Psi^*}}(z) \leq \epsilon_{plan}$ for all $z \in \mathcal{Z}_{\mathcal{F}}$, where $\epsilon_{plan} \geq 0$.*

The policy executed by IFBA in $\mathcal{G}$ is $\pi_F(s) = \pi^\circ_{M_{\Psi^*}}(\Psi^*(s))$.

2.3 Performance Guarantees (Motivational)

We first establish a performance guarantee for the IFBA, which operates with a perfect abstract model.

Theorem 2.4 (Performance of IFBA with Perfect Abstract Model)**.** *Let π_F be the policy derived by the Idealized Fact-Based Agent (IFBA) as defined above. If the ideal fact abstraction Ψ^* establishes an ϵ_{sim}-approximate bisimulation between $\mathcal{G}$ and M_{Ψ^*}, and the planner for M_{Ψ^*} is ϵ_{plan}-optimal, then for any state $s \in \mathcal{S}$:*

$$V^*_{\mathcal{G}}(s) - V^{\pi_F}_{\mathcal{G}}(s) \leq \frac{2\epsilon_{sim}}{1 - \gamma} + \epsilon_{plan} \tag{2}$$

Proof. We defer the proof to Appendix E. □

Performance with a Learned Abstract Model: In practice, an agent learns an *approximate* abstract model $\tilde{M}_\Psi = (\mathcal{Z}_\mathcal{F}, \mathcal{A}, \tilde{\mathcal{T}}_\Psi, \tilde{R}_\Psi, \gamma)$ from data, based on an abstraction Ψ (which itself has an associated ϵ_{sim} quality). Let π_L be an ϵ_{plan}-optimal policy for this learned model $\tilde{M}_\Psi$. The value loss is decomposed as:

$$V^*_\mathcal{G}(s) - V^{\pi_L}_\mathcal{G}(s) = \underbrace{(V^*_\mathcal{G}(s) - V^*_{\tilde{M}_\Psi}(\Psi(s)))}_{\text{Term A}} + \tag{3}$$

$$\underbrace{(V^*_{\tilde{M}_\Psi}(\Psi(s)) - V^{\pi_L}_{\tilde{M}_\Psi}(\Psi(s)))}_{\text{Term B}} + \tag{4}$$

$$\underbrace{(V^{\pi_L}_{\tilde{M}_\Psi}(\Psi(s)) - V^{\pi_L}_\mathcal{G}(s))}_{\text{Term C}} \tag{5}$$

where $V^*_{\tilde{M}_\Psi}$ is the optimal value function in the learned abstract model $\tilde{M}_\Psi$.

- **Term A** ($|V^*_\mathcal{G} - V^*_{\tilde{M}_\Psi}|$): This gap comprises two parts: (1) the inherent loss from abstraction, $|V^*_\mathcal{G}(s) - V^*_{M_\Psi}(\Psi(s))| \leq \frac{\epsilon_{\text{sim}}}{1-\gamma}$, and (2) the error in the optimal value due to inaccuracies in the learned model $\tilde{M}_\Psi$ compared to the true abstract model M_Ψ, $|V^*_{M_\Psi}(\Psi(s)) - V^*_{\tilde{M}_\Psi}(\Psi(s))|$. This second part is typically bounded by $C_1 \frac{\delta_{\text{model}}}{(1-\gamma)^2}$, where δ_{model} represents a composite one-step model error (in abstract transitions and rewards) of $\tilde{M}_\Psi$ w.r.t. M_Ψ (Strehl et al., 2009; Jiang et al., 2015). Thus, Term A $\lesssim \frac{\epsilon_{\text{sim}}}{1-\gamma} + C_1 \frac{\delta_{\text{model}}}{(1-\gamma)^2}$.

- **Term B** ($V^*_{\tilde{M}_\Psi} - V^{\pi_L}_{\tilde{M}_\Psi}$): This is the planning error within the agent's learned model $\tilde{M}_\Psi$, bounded by ϵ_{plan}.

- **Term C** ($|V^{\pi_L}_{\tilde{M}_\Psi} - V^{\pi_L}_\mathcal{G}|$): This simulation error is $|V^{\pi_L}_{\tilde{M}_\Psi}(\Psi(s)) - V^{\pi_L}_\mathcal{G}(s)| \leq |V^{\pi_L}_{\tilde{M}_\Psi}(\Psi(s)) - V^{\pi_L}_{M_\Psi}(\Psi(s))| + |V^{\pi_L}_{M_\Psi}(\Psi(s)) - V^{\pi_L}_\mathcal{G}(s)|$. The first part is bounded by $C_2 \frac{\delta_{\text{model}}}{(1-\gamma)^2}$ (Simulation Lemma type result (Kearns and Singh, 2002)), and the second by $\frac{\epsilon_{\text{sim}}}{1-\gamma}$ (abstraction quality for policy π_L). Thus, Term C $\lesssim \frac{\epsilon_{\text{sim}}}{1-\gamma} + C_2 \frac{\delta_{\text{model}}}{(1-\gamma)^2}$.

Summing these bounds, the total value loss is:

$$V^*_\mathcal{G}(s) - V^{\pi_L}_\mathcal{G}(s) \lesssim \frac{2\epsilon_{\text{sim}}}{1-\gamma} + \epsilon_{\text{plan}} + (C_1 + C_2)\frac{\delta_{\text{model}}}{(1-\gamma)^2} \tag{6}$$

As above, we use these bounds only as design motivation; we do not estimate or optimize ϵ_{sim}, δ_{model}, or ϵ_{plan} (Strehl et al., 2009).

2.4 Discussion: Connecting to LLM-Based Agents

This framework motivates two LLM roles:

- **LLM as Fact Extractor (f_θ):** Approximates the abstraction Ψ by turning trajectories into concise, task-relevant atomic facts F_t that preserve value-critical structure, thereby reducing abstraction error ϵ_{sim}.
- **LLM as Latent World Model and Value Estimator (g_ϕ):** Conditioned on (o_t, F_t), simulates outcomes and estimates values, implicitly defining the learned abstract model $\tilde{M}_\Psi$; higher simulation fidelity lowers δ_{model}.

Supplying specific, missing facts leverages the LLM's prior knowledge while grounding predictions, aiming to reduce aliasing (proxy ϵ_{sim}) and one-step error (proxy δ_{model}), which can in turn lower planning suboptimality ϵ_{plan} (Eq. (6)). Facts are learned online, purely in-context, and kept *minimal*: proposed facts pass a *predictive-consistency filter*—we replay the just-finished episode while conditioning the simulator (temperature 0) on the candidate fact and retain it only if the held-out next-step prediction error decreases. This is a heuristic, post-hoc, model-internal check (hindsight), not a causal test; a stronger trajectory-grounded validation variant is possible and detailed in Appendix N.1. Environment-level validation is left to future work, while the lightweight deployed check still helps curb hallucinations and prune stale knowledge. Minimal, relevant facts also keep prompts focused, avoiding dilution within the context window. The next section instantiates these principles: a depth-limited lookahead uses h_ω for proposal, simulation, and valuation, all conditioned on (o_t, F_t) within a learned, fact-based abstract MDP.

3 METHOD: LLM AGENT WITH ATOMIC FACT AUGMENTATION AND LOOKAHEAD PLANNING

Our proposed agent, the LLM-based World Model Planning Agent (LWM-Planner), enhances its decision-making capabilities through a synergistic combination of online atomic fact learning and LLM-driven lookahead search. The overarching goal is to enable the agent to learn from its interactive experiences entirely in-context, without any updates to the underlying LLM weights, and to leverage this learned knowledge to improve its planning and achieve more optimal behavior. The agent's architecture and operation can be understood through two main interacting processes: the

dynamic management of atomic facts and the lookahead planning mechanism that utilizes these facts. A high-level summary of the agent's operational cycle is provided in Algorithm 1, detailed in Section D. The overarching goal is to enable the agent to learn from its interactive experiences entirely in-context and to employ the lookahead planning mechanism that utilizes these facts to approximate strong actions (improving ϵ_{plan} via better rollouts), while fact-conditioning targets lower one-step error (proxy δ_{model}).

The LWM-Planner maintains a concise representation of its world understanding and recent interactions. Core to its state are a short-term **interaction history** ('history'), which is a deque of recent observation-action pairs, and a longer-term, distilled knowledge base in the form of an **atomic fact set** ('facts'). These facts serve as the cornerstone of the agent's learned abstraction; they are intended to capture the most salient, value-relevant aspects of the environment discovered through experience. The set is therefore composed of textual statements (e.g., "object X is on table Y," "door Z is locked") that are crucial for task completion. These, along with an environment description and a list of allowed actions, provide the necessary context for the LLM components.

The process of learning and refining the atomic fact set is central to the agent's adaptability. This occurs primarily at the end of each episode through a "reflection" phase. After an episode concludes, the complete trajectory of observations, actions, rewards, and outcomes is provided to an LLM. This LLM is tasked with identifying and generating "minimal new atomic facts" that were not previously known (i.e., not in the current `factsset`) but are *deemed critical for better predicting state values or rewards in the future*. The aim is to distill the most salient pieces of information from the recent experience that, if known earlier, could have led to improved decision-making. Candidate facts are therefore subjected to a *predictive-consistency filter*: we replay the just-finished episode while conditioning the simulator (temperature 0) on the candidate fact and accept it only if the *held-out next-step* prediction error (reward, termination, and/or next observation) decreases using cached model calls. This is a heuristic, post-hoc check performed on the same episode and is model-internal rather than ground truth; a stronger trajectory-grounded variant is given in Appendix N.1; it is *not* a causal test. As a sanity check, we discard any fact that is later contradicted by observations in the following episode. We leave environment-level validation and cross-episode checks to future work. This aims to reduce state aliasing (proxy ϵ_{sim}) by enriching $z_t = (o_t, F_t)$ with information that better separates states with different values or optimal actions. These newly extracted and validated facts are then added to the `facts` deque. To maintain the conciseness and relevance of this knowledge base, an optional fact compression step can be performed. Here, an LLM reviews the entire set of current facts and attempts to eliminate redundancies or overly specific information, producing a more compact yet informationally rich set of facts. This evolving `factsset` serves as a dynamically updated, experience-grounded augmentation for all subsequent LLM reasoning during planning. We cap the fact memory by context length; older facts are dropped by redundancy before recency.

Decision-making in LWM-Planner is orchestrated by recursive lookahead search. This search is bounded by a configurable depth (e.g., $d = 3$) and branching factor (e.g., $b = 4$). To ensure deterministic planning behavior within a single search instance, all LLM calls during this phase operate with a temperature of zero. A small step penalty is also incorporated to favor more efficient solutions. The lookahead search relies on three specialized LLM-driven functionalities, invoked via structured function calls: `propose_actions` for suggesting likely candidate actions from a given state; `simulate_step` , which acts as the *latent* world model $\tilde{T}_\Psi, \tilde{R}_\Psi$. Conditioned on $z_t = (o_t, F_t)$ and a proposed action a_i, it predicts the next (potentially latent) observation o', immediate reward r', and termination status. The accuracy of this LLM-based simulation, enhanced by the atomic facts, targets lower one-step prediction error (proxy δ_{model}). Finally, `estimate_value` approximates the value function $\hat{V}_{\tilde{M}_\Psi}$ of the learned abstract MDP. It assesses the long-term utility of states, particularly those at the frontier of the search (leaf nodes or terminal states). Facts also help ground this estimation (e.g., proximity to a known goal or hazard fact).

The planning process begins at the current observation o_t. First, the action proposal LLM, conditioned on o_t, the interaction history, and the current atomic facts, suggests a set of candidate actions. For each proposed action a_i, an estimated Q-value, $Q(o_t, a_i)$, is computed. This computation involves invoking the simulation LLM (again, conditioned on the current state, action, history, and facts) to predict the immediate reward r' and next (potentially latent) observation o'. If o' is a terminal state or the maximum search depth is reached, the value estimation LLM is called to predict the future cumulative reward from o'. Otherwise, the search recurses from o' with decremented depth. The Q-value is then a combination of the immediate simulated reward and the discounted value of the subsequent state, plus any step penalties $Q(o_t, a_i) = r' - \lambda_{\text{step}} + \gamma \hat{V}(o')$. After evaluating all initial candidate actions, the action yielding the highest Q-value is selected for execution in the environment. To manage computational overhead

during a single planning phase, the results of these LLM calls (proposal, simulation, and value estimation) are memoized based on their inputs.

4 RELATED WORK

Our work builds upon several lines of research in LLM-based agents, model-based reinforcement learning, and the use of external knowledge for planning—see Appendix A for an extended related work. Unlike ToT/RAP, which expand search over raw trajectories, our agent learns a compact, symbolic memory online and uses it to condition the simulator itself, improving rollouts at the same test-time budget.

LLM Agents Early LLM agents like ReAct (Yao et al., 2023) introduced the concept of interleaving reasoning (thought) and action generation. Reflexion (Shinn et al., 2023) extended this by incorporating self-reflection, where an LLM analyzes past failures to generate textual feedback for future trials. This episodic learning is akin to our fact extraction, but Reflexion focuses on high-level advice rather than structured atomic facts for a world model. Many agents operate in a model-free manner or rely heavily on in-context exemplars from fixed datasets.

LLM-Based Planning and World Models Several approaches have explored using LLMs for planning. Some use LLMs to score or propose actions within classical search algorithms like Monte-Carlo Tree Search (MCTS) (Hao et al., 2023; Kagaya et al., 2024; Liu et al., 2024) or to expand deliberate search trees as in Tree-of-Thought (ToT) (Yao et al., 2024). Retrieval-Augmented Planning (RAP) (Kagaya et al., 2024) retrieves full past trajectories to inform MCTS, often requiring environment interaction for tree expansion, while ExpeL (Zhao et al., 2024) distils salient information from offline experience buffers to guide planning at test time. Other works like (Chae et al., 2024) use LLMs to build explicit, but often one-step, world models that predict state transitions or webpage changes, and (Xie et al., 2023) translate natural language goals into formal planning problems. Our approach differs by using the LLM itself as a latent world model for multi-step simulation during lookahead, conditioned on dynamically extracted atomic facts that are validated against experience. This enables online abstraction learning rather than depending on pre-collected trajectory stores and provides concise, symbolic grounding that directly feeds the planner. The idea of an LLM as a "simulator" has been explored, e.g., for few-shot generation (Prystawski et al., 2023), but its integration with online fact-based learning for improved planning is a novel aspect of our work.

Dyna-Style Architectures and Fact-Based RL Our method is inspired by Dyna-style reinforcement learning (Sutton, 1990; Sutton and Barto, 2018), where an agent learns a model of the world from real interactions and then uses this model to generate simulated experiences for planning. In our case, the "model" is implicitly represented by the set of atomic facts combined with the LLM's inherent simulation capabilities. The extraction of facts from trajectories is analogous to model learning, and the lookahead search is planning with this model. While traditional Dyna uses tabular or parametric models, we leverage the LLM's ability to reason over textual facts. The concept of using facts or symbolic knowledge in RL is not new (Abel et al., 2020), but its integration with LLM-driven simulation and online fact extraction is a key aspect of our work.

Knowledge Augmentation for LLMs Retrieval-Augmented Generation (RAG) (Lewis et al., 2021) is a common paradigm for providing LLMs with external knowledge. Our fact extraction and augmentation mechanism can be seen as a specialized form of RAG where the "retrieved" knowledge (atomic facts) is actively generated and refined from the agent's own experience rather than drawn from a static corpus. This makes the knowledge highly task-specific and current, directly addressing the information needs identified through interaction, rather than relying on potentially less relevant or outdated general knowledge.

Our work distinguishes itself by the tight integration of online atomic fact learning from episodic experience with an LLM-driven, multi-step lookahead planner where the LLM serves as both a latent world model and value function, all operating in-context without weight updates. This focus on distilled, symbolic knowledge (atomic facts) aims to provide a more structured and efficient way for the LLM to learn from experience compared to methods relying on raw trajectory retrieval or general textual reflections.

5 EXPERIMENTS

We evaluate our LWM-Planner to assess its ability to learn from experience and improve its decision-making accuracy and task performance over time. We evaluate behavioural improvements only; ϵ_{sim}, δ_{model}, and ϵ_{plan} are used only as motivating proxies, and we do not claim formal guarantees.

Benchmark Environments: We use three different diverse environment domains, from which we can procedurally generate a near limitless amount of different environments. First, we create a procedurally generated text version of the classic environment of Frozen Lake (Brockman et al., 2016), where we can alter the prob-

Table 1: Cumulative return (**higher better**); mean ± 95% CI, for each benchmark method across each environment. Results are averaged over ten random seeds. LWM-Planner consistently delivers the highest return, indicating more efficient task completion than the non-symbolic search baselines.

Method (metric)	TextFrozenLake (4×4; h=0.9)	CrafterMini (5×5)	ALFWorld-A	ALFWorld-B	ALFWorld-C
LWM-Planner (Cum. return ↑)	**31.80±20.39**	**150.30±44.94**	**21.33±9.53**	**22.89±12.11**	**19.50±8.37**
ReAct + FEC (Cum. return ↑)	20.20±12.19	149.70±55.50	4.70±2.46	15.50±6.51	10.60±3.71
ReAct (Cum. return ↑)	−265.20 ± 33.59	92.00±57.16	12.60±0.37	12.80±0.45	12.50±0.38
Reflexion (Cum. return ↑)	−61.10±4.80	87.20±51.45	11.00±0.00	11.00±0.45	11.33±0.38
RAP (Cum. return ↑)	−83.25±20.92	0.80±69.02	5.60±2.42	17.80±1.84	14.20±2.69
ToT (Cum. return ↑)	−56.20±4.84	51.40±118.56	1.00±0.88	2.80±0.56	5.20±1.84
Random (Cum. return ↑)	−80.00±4.49	−289.00±8.56	0.00±0.00	0.00±0.00	0.00±0.00

ability that all the tiles are holes (h) and ensure that each board is always at least solvable. Moreover, we use the standard ALFWorld environments (Shridhar et al., 2020b) (randomly sampling three environments in the main paper, with more in the appendix). ALFWorld is a text environment that parallels embodied worlds in the ALFRED dataset (Shridhar et al., 2020a), where each environment requires agents to reason to solve embodied tasks in a home environment. Furthermore, we benchmark against CrafterMini, a procedurally generated mini version of Crafter (Hafner, 2021)—a 2D world where the player needs to explore for resources, collect materials and build tools. The goal here is to craft an iron pickaxe, which can only be done by crafting two previous items and using collected resources. We detail all environments in Appendix F.

Benchmark Methods: We seek to provide competitive benchmarks; therefore, we compare against **ReAct** (Yao et al., 2023), reasoning then acting, which observes the current observation, interaction history, and the environment description. Building on top of ReAct, we compare with **Reflexion** (Shinn et al., 2023), which maintains a buffer of previously learned verbal lessons on how to act better that is appended to the end of each episode, and is included in the agent's prompt. Moreover, we consider two strong search-based planners, Tree-of-Thought (**ToT**) (Yao et al., 2024) and Retrieval-Augmented Planning (**RAP**) (Kagaya et al., 2024), both of which expand lookahead trees using LLM calls but do not learn symbolic abstractions. We include an ablation of our method, **ReAct + FEC**, which is a ReAct agent with Fact Extraction and Compression as done in LWM-Planner but without the tree search, our full **LWM-Planner**, and a **Random** policy. We provide full benchmark method details in Appendix G.

Evaluation: We run each LLM Agent method for 300 environment steps unless otherwise noted, tracking episode return and the number of steps taken for each episode. After 300 steps, we compute the cumulative return/total reward (sum of returns up to 300 steps) and report the mean ± 95% confidence interval over three random seeds. We report mean±95% CI over three seeds due to inference-time cost. To enable replication we provide detailed implementation and evaluation details in the appendix. Further experimental setup and evaluation details, including the precise success criteria and discussion of the 300-step interaction budget, are given in Appendix H. A quantitative analysis of computational cost is provided in Appendix J.6. Additional robustness results, including generalization to a smaller backbone, a compute-performance Pareto analysis, and an adversarial filter audit, are reported in Appendix J.7, Appendix J.8, and Appendix J.9.

5.1 Main Results

We evaluated all the LLM benchmark methods across all environments and tabulate the results in Table 1. LWM-Planner on average achieves higher cumulative return on every complex multi-step environment, reflecting that fact-augmented planning finds substantially more efficient solutions than competing approaches. Search-only baselines (ToT, RAP) are weaker at the same test-time budget, consistent with the value of a learned symbolic abstraction for guiding lookahead. The ReAct + FEC ablation confirms that fact learning alone provides a large benefit, while the full LWM-Planner combines this with grounded lookahead to deliver the strongest results.

Finally, Table 10 summarises targeted ablations showing that unguided lookahead can even harm performance, whereas grounding the LLM world model with learned atomic facts is the dominant contributor to the observed gains. Overall, the results suggest that additional test-time search helps most when it is guided by compact learned facts rather than applied in an unguided manner.

Table 2: **Comparison of increasing environment state-action space.** Normalised cumulative return (**higher better**) and steps per success (**lower better**); mean ± 95% CI, for each benchmark method across each environment. LWM-Planner performs the best across all environments. Results are averaged over three random seeds. Normalised cumulative return is the cumulative return normalized to be between 0 and 100, where 0 corresponds to Random and 100 to the best score among all methods on that environment.

Method (metric)	TextFrozenLake (4×4; h=0.9)	TextFrozenLake (6×6; h=9)	TextFrozenLake (8×8; h=5)
LWM-Planner (Cum. return norm ↑)	**100.00±46.34**	**100.00±30.91**	**100.00±90.33**
(Steps/Success ↓)	**6.00±0.00**	**13.67±11.47**	**42.83±80.65**
ReAct + FEC (Cum. return norm ↑)	85.90±31.81	90.87±22.45	18.18±156.46
(Steps/Success ↓)	**6.00±0.00**	77.33±199.12	–
ReAct (Cum. return norm ↑)	−114.10 ± 11.03	−279.57 ± 8.57	−336.36 ± 22.58
(Steps/Success ↓)	–	–	–
Reflexion (Cum. return norm ↑)	17.31±13.16	47.83±14.12	−212.12 ± 104.31
(Steps/Success ↓)	23.33±7.99	–	–
Random (Cum. return norm ↑)	0.00±0.00	0.00±0.00	0.00±0.00
(Steps/Success ↓)	–	–	–

For completeness, the full ALFWorld 134-task results—including per-task cumulative returns—are provided in Appendix J.1.

5.2 Insight Experiments

In the following, we gain insight into why LWM-Planner outperforms Reflexion and ReAct.

How does LWM-Planner with its atomic fact learning lead to a higher cumulative return over time compared to baselines? To investigate why LWM-Planner achieves a higher cumulative return, as seen in Table 1, we can qualitatively investigate the facts that it learns throughout its process. Specifically on TextFrozenLake ($4 \times 4; h = 0.9$), it learns the hole locations through trial and error initially, as a hole terminates an episode, then during its fact extraction stage it self extracts facts that if known earlier would have improved its value predictions. The agent therefore learns atomic concise facts describing where the holes in the state are, allowing it to avoid them. We provide detailed facts that are retrieved as a case study in Appendix I.

Moreover, while capable, ReAct struggled with long-horizon planning and adapting to subtle but critical state changes that weren't immediately obvious from the current observation alone. Whereas, Reflexion, showed learning by refining its high-level strategy. Such benchmark trajectories and memories are outlined in Appendix I.

Can LWM-Planner scale better with increasing state-action space? To investigate this we used our procedurally generated TextFrozenLake environments to generate environments of increasing board size, and hence state-action space. We tabulate these results for all the benchmarks in Table 2. Interestingly, LWM-Planner achieves the highest normalised cumulative return, and crucially, as the state-action space increases, the other baselines degrade; whereas LWM-Planner is still able to online-learn and solve the environment. This verifies that the combination of both having the fact extraction and compression, plus the ability to forward plan, is crucial, and provides an effective in-context online learning method to learn the minimal fact representations of the environment, in this case, where the holes are to solve the environment optimally.

6 CONCLUSION

Compact, experience-derived facts can make test-time lookahead more useful for LLM agents in partially observable, multi-step environments. LWM-Planner implements this idea by extracting atomic facts from episodic experience and using them to augment action proposal, latent world model simulation, and state-value estimation within a recursive lookahead search.

Our approach allows the agent to distill missing task information from interaction and feed it back into future reasoning, leading to more accurate simulations and value assessments within its lookahead search. This occurs without any LLM fine-tuning, relying entirely on in-context learning augmented by dynamically generated facts. We provided a theoretical motivation for this fact-based approach, linking agent performance to the quality of the learned factual abstraction and the fidelity of fact-conditioned simulation.

Across our benchmark suite, empirical evaluations show that LWM-Planner learns from experience and improves cumulative return over baselines that lack this focused fact-learning mechanism or grounded looka-

head. The key lies in providing the LLM with the specific, missing pieces of information (atomic facts) it needs to better ground its generative and reasoning capabilities in the context of the current task. More broadly, these results suggest that a small learned symbolic memory can serve as an effective interface between in-context experience and test-time planning.

Future work includes exploring more sophisticated fact extraction and management techniques, such as leveraging insights from causal discovery to identify truly influential facts; dynamically adjusting search depth based on task complexity or uncertainty; and investigating methods for the agent to explicitly identify when its factual knowledge is insufficient and trigger targeted exploration. The LWM-Planner represents a step towards more robust, adaptive, and experience-grounded LLM agents for complex sequential decision-making.

Acknowledgements

We extend our gratitude to the anonymous reviewers, area and program chairs, and members of the van der Schaar lab for their valuable feedback and suggestions. SH, ML & TP gratefully acknowledge the sponsorship and support of AstraZeneca. This work was supported by Azure sponsorship credits granted by Microsoft's AI for Good Research Lab and by Microsoft's Accelerate Foundation Models Academic Research Initiative.

References

David Abel, Nate Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael Littman. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, pages 1639–1650. PMLR, 2020.

Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=HyxQzBceg.

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

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. *arXiv preprint arXiv:2410.13232*, 2024.

Ruomeng Ding, Wei Cheng, Minglai Shao, and Chen Zhao. Skillgen: Learning domain skills for in-context sequential decision making. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 30512–30520, 2026.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, et al. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128, 2024.

Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *UAI*, volume 4, pages 162–169, 2004.

Isabel O Gallegos, Ryan A Rossi, Joe Barrow, Md Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K Ahmed. Bias and fairness in large language models: A survey. *Computational Linguistics*, 50(3):1097–1179, 2024.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

Paul Hager, Friederike Jungmann, Robbie Holland, Kunal Bhagat, Inga Hubrecht, Manuel Knauer, Jakob Vielhauer, Marcus Makowski, Rickmer Braren, Georgios Kaissis, et al. Evaluation and mitigation of the limitations of large language models in clinical decision-making. *Nature medicine*, 30(9):2613–2622, 2024.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=VTWWvYtF1R.

Samuel Holt, Zhaozhi Qian, Tennison Liu, James Weatherall, and Mihaela van der Schaar. Data-driven discovery of dynamical systems in pharmacology using large language models. *Advances in Neural Information Processing Systems*, 37:96325–96366, 2024.

Samuel Holt, Todor Davchev, Dhruva Tirumala, Ben Moran, Atil Iscen, Antoine Laurens, Yixin Lin, Erik Frey, Markus Wulfmeier, Francesco Romano, et al. Evocontrol: Multi-frequency bi-level control for high-frequency continuous control. In *International Conference on Machine Learning*, pages 23451–23512. PMLR, 2025a.

Samuel Holt, Max Ruiz Luyten, Antonin Berthon, and Mihaela Van Der Schaar. G-sim: Generative simulations with large language models and gradient-free calibration. In *International Conference on Machine Learning*, pages 23513–23561. PMLR, 2025b.

Nan Jiang, Alex Kulesza, and Satinder Singh. Abstraction selection in model-based reinforcement learning. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Confer-*

ence on Machine Learning, volume 37 of *Proceedings of Machine Learning Research*, pages 179–188, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/jiang15.html.

Krzysztof Kacprzyk, Samuel Holt, Jeroen Berrevoets, Zhaozhi Qian, and Mihaela van der Schaar. ODE discovery for longitudinal heterogeneous treatment effects inference. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=pxI5IPeWgW.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998. ISSN 0004-3702. doi: 10.1016/S0004-3702(98)00023-X.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. RAP: Retrieval-augmented planning with contextual memory for multimodal LLM agents. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024. URL https://openreview.net/forum?id=Xf49Dpxuox.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49:209–232, 2002.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL https://arxiv.org/abs/2005.11401.

Kun Li, Tingzhang Zhao, Wei Zhou, and Songlin Hu. Dora: Dynamic optimization prompt for continuous reflection of llm-based agent. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7546–7557, 2025.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for mdps. In *International Symposium on Artificial Intelligence and Mathematics, AI&Math 2006, Fort Lauderdale, Florida, USA, January 4-6, 2006*, 2006. URL http://anytime.cs.umass.edu/aimath06/proceedings/P21.pdf.

Jonathan Light, Wei Cheng, Benjamin Riviere, Wu Yue, Masafumi Oyamada, Mengdi Wang, Yisong Yue, Santiago Paternain, and Haifeng Chen. Disc: Dynamic decomposition improves llm inference scaling. *arXiv preprint arXiv:2502.16706*, 2025a.

Jonathan Light, Yue Wu, Yiyou Sun, Wenchao Yu, Yanchi Liu, Xujiang Zhao, Ziniu Hu, Haifeng Chen, and Wei Cheng. Sfs: Smarter code space search improves llm inference scaling. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency, 2024. URL https://arxiv.org/abs/2309.17382.

Judea Pearl. *Causality*. Cambridge University Press, 2 edition, 2009.

Thomas Pouplin, Hao Sun, Samuel Holt, and Mihaela Van der Schaar. Retrieval-augmented thought process as sequential decision making. *arXiv e-prints*, pages arXiv–2402, 2024.

Ben Prystawski, Michael Y. Li, and Noah D. Goodman. Why think step by step? reasoning emerges from the locality of experience, 2023. URL https://arxiv.org/abs/2304.03843.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*, 2024.

Balaraman Ravindran and Andrew G. Barto. *An algebraic approach to abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 2004. AAI3118325.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020a.

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

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite mdps: Pac analysis. *J. Mach. Learn. Res.*, 10:2413–2444, December 2009. ISSN 1532-4435.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, 1990.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

Xinglin Wang, Yiwei Li, Shaoxiong Feng, Peiwen Yuan, Yueqi Zhang, Jiayi Shi, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. Every rollout counts: Optimal resource allocation for efficient test-time scaling. *arXiv preprint arXiv:2506.15707*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Qiyao Wei, Samuel Holt, Jing Yang, Markus Wulfmeier, and Mihaela van der Schaar. The ai imperative: Scaling high-quality peer review in machine learning. *arXiv preprint arXiv:2506.08134*, 2025.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models, 2023. URL https://arxiv.org/abs/2302.05128.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents, 2025. URL https://arxiv.org/abs/2410.13825.

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 (ICLR)*, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao Chen, Nan Duan Yu, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2024.

Tianjun Zhang, Aman Madaan, Luyu Gao, Steven Zheng, Swaroop Mishra, Yiming Yang, Niket Tandon, and Uri Alon. In-context principle learning from mistakes. *arXiv preprint arXiv:2402.05403*, 2024.

Andrew Zhao, Samy Badreddine, Ishan Ilanchezian, Federico Bianchi, Kyle Richardson, Rick Chen, Daniel Khashabi, Chelsea Finn, Amir Zamir, and Sebastian Ruder. ExpeL: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18900–18908, 2024.

Eric Zhao, Pranjal Awasthi, and Sreenivas Gollapudi. Sample, scrutinize and scale: Effective inference-time search by scaling verification. *arXiv preprint arXiv:2502.01839*, 2025.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. *arXiv preprint arXiv:2306.07863*, 2023.

Ruiwen Zhou, Yingxuan Yang, Muning Wen, Ying Wen, Wenhao Wang, Chunling Xi, Guoqiang Xu, Yong Yu, and Weinan Zhang. Trad: Enhancing llm agents with step-wise thought retrieval and aligned decision. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–13, 2024.

CHECKLIST

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes — Sections 2 and 3]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes — Section 2 (sample guarantees) and Appendix J.6 (computational cost)]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [No — We provide an extensive appendix which includes implementation details.]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes — Section 2]
   (b) Complete proofs of all theoretical results. [Yes — Appendix E]
   (c) Clear explanations of any assumptions. [Yes — Section 2]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a

URL). [Yes — The environments are publicly available, and implementation details are described in Appendix H.]

(b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes — Appendix H]

(c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes — Section 5 and Appendix H]

(d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes — Appendix H]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:

(a) Citations of the creator If your work uses existing assets. [Yes — Section F]

(b) The license information of the assets, if applicable. [No — Public benchmarks cited without explicit license discussion]

(c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]

(d) Information about consent from data providers/curators. [Not Applicable]

(e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

(a) The full text of instructions given to participants and screenshots. [Not Applicable]

(b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

(c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Appendix

Table of Contents

A Extended Related Work

Our work builds upon several lines of research in LLM-based agents, model-based reinforcement learning, and the use of external knowledge for planning. The related work in the following extends that given in the main paper.

LLM Agents Early LLM agents like ReAct (Yao et al., 2023) introduced the concept of interleaving reasoning (thought) and action generation. Reflexion (Shinn et al., 2023) extended this by incorporating self-reflection, where an LLM analyzes past failures to generate textual feedback for future trials. This episodic learning is akin to our fact extraction, but Reflexion focuses on high-level advice rather than structured atomic facts for a world model. More recent agent work continues to target the same interactive reasoning bottlenecks with stronger scaffolding: Agent Q combines guided MCTS, self-critique, and offline preference optimization for web agents (Putta et al., 2024), while DORA studies how to sustain iterative reflection in MiniWoB++ and ALFWorld by dynamically optimizing reflection prompts (Li et al., 2025). For our purposes, ReAct and Reflexion remain the most comparable training-free baselines, since they isolate prompt-level acting and reflection without extra policy learning or auxiliary prompt-optimization modules.

Experience Retrieval and Skill Abstraction in POMDPs Several recent methods improve in-context sequential decision making by retrieving or synthesizing better contextual guidance from prior experience. Synapse retrieves abstracted full trajectories as exemplars for computer-control tasks (Zheng et al., 2023), while TRAD performs step-wise thought retrieval and aligned local-context selection to reduce irrelevant prompt content in ALFWorld- and Mind2Web-style settings (Zhou et al., 2024). A complementary direction is to distill higher-level reusable guidance: LEAP learns explicit principles from mistakes for few-shot adaptation (Zhang et al., 2024), and SkillGen constructs domain-level action-centric skills to produce focused, step-wise prompts for sequential decision making (Ding et al., 2026). Our approach differs from both families. We do not retrieve raw trajectories from a memory bank or precompute domain skills offline; instead, we extract minimal atomic facts online from the agent's own episodes and use them to ground a latent world-model lookahead in the current environment's transition structure.

LLM-Based Planning and World Models Several approaches have explored using LLMs for planning. Some use LLMs to score or propose actions within classical search algorithms like Monte-Carlo Tree Search (MCTS) (Hao et al., 2023; Kagaya et al., 2024; Liu et al., 2024). For instance, Retrieval-Augmented Planning (RAP) (Kagaya et al., 2024; Pouplin et al., 2024) retrieves full past trajectories to inform MCTS, often requiring environment interaction for tree expansion. Other works like (Chae et al., 2024) use LLMs to build explicit, but often one-step, world models that predict state transitions or webpage changes. (Xie et al., 2023) use LLMs to translate natural language goals into formal planning problems. A related but orthogonal direction uses LLMs to design or refine the simulator itself rather than to serve as the online planner: G-Sim combines LLM-guided structural design with empirical calibration for general-purpose simulators (Holt et al., 2025b), while D3 uses LLMs to iteratively propose, evaluate, and refine interpretable dynamical-system models from data (Holt et al., 2024). Our approach differs by using the LLM itself as a latent world model for multi-step simulation during lookahead, conditioned on dynamically extracted atomic facts, rather than just retrieving raw trajectories or learning a separate explicit model. The idea of an LLM as a "simulator" has been explored, e.g., for few-shot generation (Prystawski et al., 2023), but its integration with online fact-based learning for improved planning is a novel aspect of our work.

Inference-Time Scaling Recent work has shown that scaling test-time compute can substantially improve static reasoning and code-generation performance, in some cases more efficiently than scaling model size alone (Snell et al., 2024; Wu et al., 2024). DISC dynamically decomposes reasoning traces during search to focus compute on difficult steps (Light et al., 2025a); SFS frames code generation as search over code space with verifier feedback (Light et al., 2025b); Sample, Scrutinize and Scale studies how sampling-based search improves as verification is scaled (Zhao et al., 2025); and Every Rollout Counts derives direction-oriented resource allocation for fixed rollout budgets in mathematical reasoning (Wang et al., 2025). These methods mostly study static tasks such as math or code, where candidate solutions can be compared, verified, or revised after the fact. Our setting is interactive and partially observable: search quality depends not only on search budget, but also on whether the agent has learned the environment-specific latent state information needed for accurate transition prediction. The Pareto analysis in Appendix J.8 complements this literature by showing that raw search depth alone saturates in our POMDP setting, whereas fact-grounded lookahead converts additional compute into better decisions.

Dyna-Style Architectures and Fact-Based RL Our method is inspired by Dyna-style reinforcement learning (Sutton, 1990; Sutton and Barto, 2018), where an agent learns a model of the world from real interactions and then uses this model to generate simulated experiences for planning. In our case, the "model" is implicitly represented by the set of atomic facts combined with the LLM's inherent simulation capabilities. The extraction of facts from trajectories is analogous to model learning, and the lookahead search is planning with this model. While traditional Dyna uses tabular or parametric models, we leverage the LLM's ability to reason over textual facts. Outside language-mediated POMDPs, adjacent continuous-control work also studies multi-timescale controller structure for efficient exploration and fast reactions, e.g. EvoControl's learned high-/low-frequency bi-level control (Holt et al., 2025a). The concept of using facts or symbolic knowledge in RL is not new (Abel et al., 2020), including outside the LLM domain, for instance, through the discovery of ODEs for treatment effect inference (Kacprzyk et al., 2024). Our integration of LLM-driven simulation with online atomic fact extraction brings this paradigm into the modern LLM agent landscape.

Knowledge Augmentation for LLMs Retrieval-Augmented Generation (RAG) (Lewis et al., 2021) is a common paradigm for providing LLMs with external knowledge. Our fact extraction and augmentation mechanism can be seen as a specialized form of RAG where the "retrieved" knowledge (atomic facts) is actively generated and refined from the agent's own experience rather than drawn from a static corpus. This makes the knowledge highly task-specific and current, directly addressing the information needs identified through interaction, rather than relying on potentially less relevant or outdated general knowledge. More broadly, LLM-based augmentation has also been proposed for scientific-evaluation workflows such as peer review (Wei et al., 2025), although that setting targets human deliberation rather than environment interaction.

Our work distinguishes itself by the tight integration of online atomic fact learning from episodic experience with an LLM-driven, multi-step lookahead planner where the LLM serves as both a latent world model and value function, all operating in-context without weight updates. This focus on distilled, symbolic knowledge (atomic facts) aims to provide a more structured and efficient way for the LLM to learn from experience compared to methods relying on raw trajectory retrieval or general textual reflections.

B Prompt Structures

This section provides the core prompt structures employed by LWM-Planner's LLM components. These conceptual prompts are dynamically populated at runtime with specific content such as environment descriptions (`{{env_description_str}}`), the current set of atomic facts (`{{current_facts_list_str}}`), the current observation (`{{current_observation_str}}`), and relevant interaction history (`{{history_lines_str}}`). All LLM interactions leverage a structured function-calling interface. The prompts guide the LLM to produce a "thought" (chain-of-thought reasoning) and then invoke a specified function with the relevant arguments, following Wei et al. (2022).

B.1 Fact Extractor LLM Prompts (f_θ)

Invoked post-episode to extract new atomic facts and refine the fact memory. These LLM calls use a temperature of 0.0 for deterministic fact processing.

B.1.1 Fact Elicitation from Trajectory (`fact_extraction`)

```
Fact Elicitation Prompt:


SYSTEM: You are an expert agent.

USER: You are a LLM fact extraction agent.  Operating in the following environment defined below.
Your task is to extract atomic facts that you did not know already to help with predicting the
next state value / next reward, such that if you had this fact you would have improved your
prediction for the next state value, when being a world model (that is be able to complete the
task optimally in the minimum number of steps, therefore extract key information that helps you).

ENVIRONMENT DESCRIPTION:

{{ env_description_str }}
```

episode_trajectory_summary_str *// This includes: Outcome: episode_outcome_str (Total Reward: ...) // And a sequence like: // "1. Obs: S00 | Act: right | Reward: 0.0 | Next_Obs: S01 // 2. Obs: S01 | Act: down | Reward: -1.0 | Next_Obs: S11 (Hole)"*

We already know and have the following facts (ensure you do not duplicate them) (at beginning of episode):

{{ current_facts_list_str }}

// e.g., ["hole@(1,1)", "object_A_is_on_table_B"]

Now respond with minimal new atomic facts (at beginning of episode) that you did not already know, for the rest of the states assume you already know them. Make facts as concise as possible. Optimize them for other agents reading and decision making given a current state. Never duplicate the facts if they already exist within our following fact set. Do not include any other text or reasoning, just the facts. If no new facts just return empty string. Use function "fact_-extraction" to do this now.

Function Call: fact_extraction Arguments: "thought": (string) Your reasoning process for identifying these new facts. "new_facts": (list of strings) The list of newly extracted atomic facts. If no new critical facts are found, provide an empty list. Example: ["hole_at(1,1)", "goal_at(3,3)"]

B.1.2 (Optional) Fact Memory Compression and Refinement (fact_redundancy_remover)

```
Fact Memory Compression Prompt:


SYSTEM: You are an expert agent.

USER: Remove any redundant facts that are already included in the list of all facts given to you.
You will also always be given the environment description, therefore you can use that to help
you remove any redundant facts.  Always keep all exhaustive factual knowledge, just remove any
duplicate facts, or redundant information already contained within the environment description.
You optimize the facts so they can be read by another LLM agent using them for being a world model
of the environment (where the agent has to simulate given a state,action to predict the next state,
next reward and terminal state).  Remove any redundancy, otherwise copy over the existing facts
verbatim.

ENVIRONMENT DESCRIPTION:

{{ env_description_str }}


Facts (at beginning of episode):

{{ current_facts_list_for_compression_str }}


// e.g., ["hole_at(1,1)", "object_A_is_on_table_B", "hole_at(row=1,col=1)"]

List of all facts (at beginning of episode) that you did not know already (not contained within
the environment description) to help with predicting the next state value / next reward, such that
if you had this fact you would have improved your prediction for the next state value, when being
a world model.  Optimize them for other agents reading and decision making given a current state.
Use function "fact_redundancy_remover" to do this now.

Function Call:  fact_redundancy_remover Arguments:  "thought":  (string) Your reasoning for the
compression and refinement decisions.  "all_facts":  (list of strings) The refined, concise list
of essential atomic facts.
```

B.2 Planner LLM Prompts (g_ϕ)

Used within the lookahead search. These LLM calls operate with a temperature of 0.0 for deterministic planning outcomes, as described in Section 3.

B.2.1 Action Proposal (propose_actions)

```
Action Proposal Prompt:

SYSTEM: You must call propose_actions.

USER: You are an next best action proposing agent, task with solving the given environment defined
below optimally.  Your task is to propose up to {{ branch_factor_int }} most likely next best
unique actions to try next that make the agent solve the environment task optimally.

Environment description:

{{ env_description_str }}

Atomic facts that help to predict next state value / next reward accurately (at beginning of
episode):

{{ current_facts_list_str }}

Current Observation:

{{ current_observation_str }}

Recent history (old->new):

{{ history_lines_str }}

// e.g., "Obs:  S00 -> Act:  right -> Obs:  S01"

You now see Observation:  {{ current_observation_str }}.  Now reason through (using the atomic
facts, and recent observation and action history), then give propose up to {{ branch_factor_int
}} most likely next best unique actions to try next that make the agent solve the environment task
optimally, each from {{ allowed_actions_list_str }}.  You will call the function propose_actions
to do this.

Function Call:  propose_actions Arguments:  "thought":  (string) Your reasoning for selecting
these actions.  "actions":  (list of strings) The proposed actions.
```

B.2.2 Latent World Model - Single Step Simulation (simulate_step)

```
World Model Simulation Prompt:

SYSTEM: You must call simulate_step.

USER: You are a latent world model for the given environment defined below. Given the current
observation and an action, predict: the next (perhaps latent) observation, immediate reward and
done flag (whether the resulting state ends the episode). You must be as accurate as possible, as
your output is used as a planner to solve the given environment optimally.

Environment description:

{{ env_description_str }}

Atomic facts that help to predict next state value / next reward accurately (at beginning of
episode):

{{ current_facts_list_str }}

Current Observation:

{{ current_observation_str }}

Recent history (old->new):

{{ history_lines_str }}

Given action to simulate the next observation and reward for:

{{ action_to_simulate_str }}

You now see Observation: {{ current_observation_str }}. Now reason through (using the atomic
facts, and recent observation and action history), and predict the next (perhaps latent)
observation, immediate reward, and done flag (whether the resulting state ends the episode) after
taking the given action of {{ action_to_simulate_str }}. You must be as accurate as possible
(for the predicted reward, and ensure your predicted next observation has enough observation
information to predict future rewards for the given task in the given environment), as your
output is used as a planner to solve the given environment optimally. You will call the function
simulate_step to do this.

Function Call: simulate_step Arguments: "thought": (string) Your reasoning for the predicted
outcome. "next_observation": (string) The predicted (perhaps latent) observation after the
action. "reward": (float) The predicted immediate reward (float) after the action. "done":
(boolean) True if the resulting state ends the episode (terminal), false otherwise.
```

B.2.3 Value Estimator (estimate_value)

```
Value Estimation Prompt:

SYSTEM: You must call estimate_value.

USER: You are a state value function estimator for the given environment defined below.  You
must predict the current cumulative future reward from the current (perhaps latent) observation.
You must be as accurate as possible, as your output is used as a planner to solve the given
environment optimally.  The environment's discount factor is {{ discount_gamma_float }}.

Environment description:

{{ env_description_str }}

Atomic facts that help to predict next state value / next reward accurately (at beginning of
episode):

{{ current_facts_list_str }}

Current Observation (to predict the current cumulative future reward for):

{{ observation_to_evaluate_str }}

Recent history (old->new):

{{ history_lines_str }}

You now see Observation:  {{ observation_to_evaluate_str }}.  Now reason through (using the atomic
facts, and recent observation and action history), and predict the current cumulative future
reward from the current (perhaps latent) observation.  You must be as accurate as possible, as
your output is used as a planner to solve the given environment optimally.  The environment's
discount factor is {{ discount_gamma_float }}.  You will call the function estimate_value to do
this.

Function Call:  estimate_value Arguments:  "thought":  (string) Your reasoning for this value
estimate.  "value":  (float) The estimated state value (float).  The cumulative future reward from
the current (perhaps latent) observation.
```

C Conceptual Details of LLM Component Prompts

This appendix provides a more formal conceptual overview of the prompts used to guide the LLM components within the LWM-Planner framework, as detailed in Section 3. These prompts are designed to elicit specific reasoning and generation capabilities from the LLMs, enabling them to function as fact extractors, world model simulators, and value function approximators, all operating through a structured function-calling interface. The specific fields like `{{ env_description_str }}`, `{{ current_facts_list_str }}`, etc., are placeholders populated dynamically by the agent at runtime.

C.1 Fact Elicitation and Memory Refinement LLM (Ψ_{LLM})

The Fact Elicitation and Memory Refinement LLM, denoted Ψ_{LLM} (see Section 2.4), is responsible for constructing and maintaining the agent's symbolic Fact Memory, $\mathcal{M}_t$. This typically occurs post-episode, leveraging the trajectory $\tau_e = (o_0, a_0, r_0, \ldots, o_H)$ from episode e. The process involves two main LLM-driven function calls: fact extraction and fact compression/refinement.

C.1.1 Fact Elicitation (`fact_extraction` call)

- **Objective**: To identify a concise set of new, task-relevant atomic facts $\Delta\mathcal{F}_e$ from the trajectory τ_e. These facts, when incorporated into the existing $\mathcal{M}_t$, are intended to improve the agent's predictive capabilities and decision-making quality, effectively learning and refining the abstraction function Ψ.
- **Input to LLM** (Context provided in the user prompt):
  1. **Environment Description** (`{{ env_description_str }}`): A comprehensive description of the environment $\mathcal{G}$, including its rules, objectives, action space $\mathcal{A}$, and the nature of observations $o \in \mathcal{O}$.
  2. **Current Fact Memory** (`{{ current_facts_list_str }}`): The set of atomic facts, $\mathcal{M}_t$, that the agent currently holds, passed as a list of strings.
  3. **Episode Trajectory Summary** (`{{ episode_trajectory_summary_str }}`): A string summarizing the completed episode τ_e, including the outcome (e.g., success/failure), total reward, and a formatted sequence of observations, actions, rewards, and next observations.
- **LLM Task Specification** (Instructions guiding the LLM to generate arguments for the `fact_extraction` function):
  1. Analyze the provided `{{ episode_trajectory_summary_str }}` in conjunction with the `{{ current_facts_list_str }}` and `{{ env_description_str }}`.
  2. Identify "minimal new atomic facts" ($\Delta\mathcal{F}_e$) that are evidenced by or can be reliably inferred from the trajectory and are *not* already present or directly implied by the `{{ current_facts_list_str }}` or `{{ env_description_str }}`.
  3. Prioritize facts crucial for explaining significant trajectory events (e.g., unexpected rewards, state transitions leading to success or failure, particularly those that would improve the prediction of state values or rewards if known beforehand).
  4. Ensure facts are concise, atomic, and adhere to any implicitly defined predicate vocabulary illustrated by examples (e.g., `hole_at(x,y)` for TextFrozenLake, `object_X_is_in_receptacle_Y` for ALFWorld).
  5. The LLM should structure its output to call the `fact_extraction` function, providing its internal reasoning as the `thought` argument and the identified new facts as a list of strings for the `new_facts` argument.
- **Qualitative Goal**: The LLM engages in a form of abductive reasoning to hypothesize underlying environmental properties or dynamics. These hypotheses, framed as new atomic facts, should explain observed phenomena in τ_e, especially aspects that were surprising or poorly modeled by the existing $\mathcal{M}_t$. The aim is to iteratively refine $\mathcal{M}_t$ towards a more accurate and value-preserving abstraction, contributing to minimizing ϵ_{sim}.

C.1.2 (Optional) Fact Compression and Refinement (`fact_redundancy_remover` call)

- **Objective**: To maintain a compact, non-redundant, and highly informative Fact Memory $\mathcal{M}_{t+1}$. This enhances computational efficiency within the LLM's context window and can improve the generalization of the Planner LLM by focusing its attention on the most salient information.

- **Input to LLM** (Context provided in the user prompt):
  1. **Environment Description** (`{{ env_description_str }}`).
  2. **Augmented Fact Set** (`{{ current_facts_list_for_compression_str }}`): The union of the previous Fact Memory and newly extracted facts, $\mathcal{M}_t \cup \Delta\mathcal{F}_e$, passed as a list of strings.
- **LLM Task Specification** (Instructions guiding the LLM to generate arguments for the `fact_redundancy_remover` function):
  1. Review the entire provided set of facts for semantic overlap, direct redundancy (e.g., facts identical to or trivially inferable from the `{{ env_description_str }}`), or subsumption by more general facts within the set.
  2. Generate a revised and refined fact set, $\mathcal{M}_{t+1}$, by removing or merging facts to enhance conciseness while preserving all critical, distinct pieces of information essential for optimal decision-making and world model accuracy.
  3. The LLM should structure its output to call the `fact_redundancy_remover` function, providing its reasoning as the `thought` argument and the complete, refined list of facts as the `all_facts` argument.
- **Qualitative Goal**: This process aims to manage the complexity of the abstract state representation $|\mathcal{Z}_\mathcal{F}|$. By ensuring $\mathcal{M}_{t+1}$ is maximally informative yet minimally redundant, it helps focus the Planner LLM's reasoning and prevents dilution of critical information, especially within a fixed context window.

C.2 Planner LLM (g_ϕ) Components for Lookahead Search

The Planner LLM, g_ϕ, is central to the lookahead search mechanism described in Section 3. It is invoked through three distinct function calls to propose actions, simulate their outcomes, and estimate the value of states encountered during the search. An abstract state z_k at any point in the search (real or simulated) is effectively represented by $(o_k, \mathcal{M}_t)$, where o_k is the observation at that point and $\mathcal{M}_t$ is the agent's current, fixed set of atomic facts for the episode.

C.2.1 Action Proposal (`propose_actions` call)

- **Objective**: To generate a focused yet diverse set of up to k_B candidate actions from the current (potentially simulated) observation o_k that are relevant for achieving the task goal or for effective exploration during planning.
- **Input to LLM** (Context provided in the user prompt):
  1. **Environment Description** (`{{ env_description_str }}`).
  2. **Current Atomic Facts** (`{{ current_facts_list_str }}`): The agent's Fact Memory, $\mathcal{M}_t$.
  3. **Current Observation** (`{{ current_observation_at_node_k_str }}`): The observation o_k from which actions are to be proposed.
  4. **Recent Trajectory History** (`{{ recent_history_for_prompt_str }}`): An excerpt of the (simulated or real) trajectory within the current lookahead search (or agent history) leading to o_k. This is typically a list of `"Obs: ..."` and `"Act: ..."` strings.
  5. **Available Actions** (`{{ available_actions_list_str }}`): The set of legally permissible actions $\mathcal{A}(s_k)$ from the underlying ground state s_k corresponding to o_k (or the full action set $\mathcal{A}$).
  6. **Branching Factor** (`{{ branch_factor_k_B_int }}`): The maximum number of actions to propose.
- **LLM Task Specification** (Instructions for the `propose_actions` function): Given o_k, $\mathcal{M}_t$, and `{{ recent_history_for_prompt_str }}`, propose up to `{{ branch_factor_k_B_int }}` distinct actions from `{{ available_actions_list_str }}` that appear most promising. The selection should be informed by the current understanding of the environment as encoded in $\mathcal{M}_t$ and the immediate context o_k. The LLM returns its reasoning (`thought`) and the list of `actions`.

C.2.2 Single-Step Abstract Simulation (`simulate_step` call)

- **Objective**: To predict the immediate outcome—next observation o'_j, immediate reward r_j, and termination status d'_j—of executing a proposed action a_j from the current observation o_k, conditioned on the Fact Memory $\mathcal{M}_t$. This approximates the abstract transition $\hat{\mathcal{T}}_\Psi$ and reward $\hat{R}_\Psi$ functions.
- **Input to LLM** (Context provided in the user prompt):
  1. **Environment Description** (`{{ env_description_str }}`).
  2. **Current Atomic Facts** (`{{ current_facts_list_str }}`, i.e., $\mathcal{M}_t$).
  3. **Current Observation** (`{{ current_observation_at_node_k_str }}`, i.e., o_k).
  4. **Action to Simulate** (`{{ action_to_simulate_str }}`, i.e., $a_j \in \mathcal{A}$).
  5. **Recent Trajectory History** (`{{ recent_history_for_prompt_str }}`) leading to o_k.
- **LLM Task Specification** (Instructions for the `simulate_step` function): Predict the `next_observation` (o'_j), `reward` (r_j), and `done` (d'_j) status that would result from taking `{{ action_to_simulate_str }}` from `{{ current_observation_at_node_k_str }}`, given `{{ current_facts_list_str }}`. The prediction should be deterministic (temperature for this LLM call is 0.0 as per Section 3) and consistent with the known facts and environment rules. The LLM returns its reasoning (`thought`) and these three predicted outcomes.
- **Qualitative Goal**: The LLM leverages $\mathcal{M}_t$ to make informed predictions. For instance, a fact like `hole_at(x,y)` should lead to a prediction of a terminal state and negative reward if a_j leads to (x, y). If facts are insufficient, the LLM relies on its pre-trained knowledge, as discussed in the context of minimizing δ_{model}.

C.2.3 Abstract State-Value Estimation (`estimate_value` call, approximating $\hat{V}_{\tilde{M}_\Psi}$)

- **Objective**: To estimate the expected total discounted future reward, $V(o_k|\mathcal{M}_t) \approx V^*_{\tilde{M}_\Psi}(z_k)$, obtainable from the abstract state $z_k \triangleq (o_k, \mathcal{M}_t)$, particularly for leaf nodes in the lookahead search tree.
- **Input to LLM** (Context provided in the user prompt):
  1. **Environment Description** (`{{ env_description_str }}`).
  2. **Current Atomic Facts** (`{{ current_facts_list_str }}`, i.e., $\mathcal{M}_t$).
  3. **Observation to Evaluate** (`{{ observation_to_evaluate_str }}`, i.e., o_k).
  4. **Recent Trajectory History** (`{{ recent_history_for_prompt_str }}`) leading to o_k.
  5. **Discount Factor** (`{{ discount_gamma_float }}`, i.e., γ).
- **LLM Task Specification** (Instructions for the `estimate_value` function): Estimate the cumulative future discounted reward (`value`) achievable from `{{ observation_to_evaluate_str }}`, considering `{{ current_facts_list_str }}` and the overall task objective. The estimation should be deterministic (temperature for this LLM call is 0.0). The LLM returns its reasoning (`thought`) and the estimated `value`.
- **Qualitative Goal**: The LLM assesses the long-term utility by considering the strategic implications of known facts (e.g., proximity to a goal, known hazards, locked doors leading to goal areas) relative to the task.

The LWM-Planner's lookahead search (Section 3) systematically invokes these LLM functionalities. The `propose_actions` function generates branches, `simulate_step` projects these branches forward one step in the abstract model $\tilde{M}_\Psi$, and `estimate_value` provides valuations $\hat{V}_{\tilde{M}_\Psi}$ at the search frontier or for terminal states. This entire process relies on the dynamically updated $\mathcal{M}_t$ to ground the LLM's powerful generative and reasoning capabilities in task-specific, experience-derived knowledge.

D Algorithm Details and Reproducibility

The LWM-Planner agent enhances its decision-making by iteratively learning atomic facts and using them in a lookahead planning process. This section details its operational cycle, broken down into a main agent loop (Algorithm 1) and sub-algorithms for the core planning (Algorithm 2) and fact learning (Algorithm 3) phases. This modular description aims to provide clarity for reproducibility, aligning with the methodology presented in Section 3 and the agent's Python implementation. Key LLM interactions are managed via a structured function-calling interface, detailed in Appendix B and Appendix C. For a compact implementation-facing specification of the core method, see Appendix D.1.

Algorithm 1: LWM-Planner: Main Agent Loop

Initialize: Global Fact Memory $\mathcal{M} \leftarrow \emptyset$;
LLM Components: Ψ_{LLM}: Fact Extractor & Refiner LLM, g_ϕ: Planner LLM, comprising $g_\phi^{\text{propose}}, g_\phi^{\text{simulate}}, g_\phi^{\text{value}}$;
Hyperparameters: $D_s, k_B, \gamma, \lambda_{\text{step}}, T_{\text{max}}, H_L$;
Short-term history buffer $\mathcal{H} \leftarrow \texttt{deque}(\text{maxlen}{=}H_L)$;

1 **for** *episode* $e \leftarrow 1$ **to** E **do**
2 $\mathcal{B}_e \leftarrow \emptyset$; `# Episode trajectory buffer for` $(o, a, r^{\texttt{real}}, o', d)$ `tuples`
3 $o_t \leftarrow$ `env.reset()`;
4 $\mathcal{H}$.clear(); $\mathcal{H}$.append(`FormatAsHistoryString("Obs:",` o_t`)`) ; `# Reset history`
5 $\mathcal{M}_{\text{current\_ep}} \leftarrow \mathcal{M}$; `# Snapshot of facts for consistent planning`
6 **for** $t \leftarrow 0$ **to** $T_{max} - 1$ **do**
7 $a_t^* \leftarrow$ `RecursiveLookaheadPlan`$(o_t, \mathcal{H}, \mathcal{M}_{\text{current\_ep}}, D_s, k_B, \gamma, \lambda_{\text{step}}, g_\phi)$;
8 Execute a_t^* in environment $\mathcal{G}$; observe real $(o_{t+1}, r_t^{\text{real}}, d_{t+1})$;
9 Add $(o_t, a_t^*, r_t^{\text{real}}, o_{t+1}, d_{t+1})$ to $\mathcal{B}_e$;
10 $\mathcal{H}$.append(`FormatAsHistoryString("Act:",` a_t^*`)`);
11 $\mathcal{H}$.append(`FormatAsHistoryString("Obs:",` o_{t+1}`)`);
12 $o_t \leftarrow o_{t+1}$;
13 **if** d_{t+1} **then**
14 **break** ; `# End episode if terminal state reached`
15 $\mathcal{M} \leftarrow$ `LearnFactsAndUpdateMemory`$(\mathcal{B}_e, \mathcal{M},$ `env_description_str`$, \Psi_{\text{LLM}})$;

Main Agent Loop (Algorithm 1) The LWM-Planner operates over a series of E episodes.

- **Initialization (Lines 1-5):** The agent starts with an empty global Fact Memory ($\mathcal{M}$). The LLM components are defined: Ψ_{LLM} for managing facts and g_ϕ for planning. The planner g_ϕ internally comprises three distinct LLM-driven functionalities: g_ϕ^{propose} for proposing actions, g_ϕ^{simulate} for simulating outcomes of actions, and g_ϕ^{value} for estimating the value of states. Hyperparameters critical for the agent's operation are set, including maximum search depth D_s, branching factor k_B, discount factor γ, step penalty λ_{step}, maximum steps per episode T_{max}, and the length H_L of the short-term interaction history buffer $\mathcal{H}$. This buffer $\mathcal{H}$ stores a rolling window of the most recent observations and actions (conceptually as formatted strings, e.g., `"Obs: <obs_string>"`) to provide immediate context to the LLMs.
- **Episodic Interaction (Lines 6-16):** For each episode:
  - An episode buffer $\mathcal{B}_e$ is initialized to log the sequence of interactions. The environment is reset, and the initial observation o_0 is used to initialize $\mathcal{H}$. A snapshot of the current global Fact Memory, $\mathcal{M}_{\text{current\_ep}}$, is taken to ensure that planning within the current episode uses a consistent set of facts learned up to that point.
  - **Per-Step Cycle (Lines 7-14):** The agent interacts with the environment step-by-step.
    - **Planning (Line 7):** The `RecursiveLookaheadPlan` sub-algorithm (Algorithm 2) is invoked. This function takes the current observation o_t, the short-term history $\mathcal{H}$, the episode's fact set $\mathcal{M}_{\text{current\_ep}}$, and planning hyperparameters to determine the best action a_t^*.

* **Interaction & Recording (Lines 8-12):** The chosen action a_t^* is executed in the actual environment $\mathcal{G}$. The resulting transition (next observation o_{t+1}, real reward r_t^{real}, and done signal d_{t+1}) is recorded in $\mathcal{B}_e$. The short-term history $\mathcal{H}$ is updated with a_t^* and o_{t+1}. The current observation o_t becomes o_{t+1}.
* **Episode Termination (Line 14):** If a terminal state is reached (d_{t+1} is true), the inner step loop concludes.

– **Fact Model Learning (Line 15):** After the episode finishes, the `LearnFactsAndUpdateMemory` sub-algorithm (Algorithm 3) is called. This function processes the trajectory $\mathcal{B}_e$ and the facts known at the start of the episode ($\mathcal{M}_{\text{current\_ep}}$) to update the global $\mathcal{M}$.

Algorithm 2: LWM-Planner: Recursive Lookahead Plan

Function `RecursiveLookaheadPlan`$(o_{curr}, \mathcal{H}_{curr}, \mathcal{M}_{ep}, D_s, k_B, \gamma, \lambda_{step}, g_\phi)$**:**
  Candidate actions $\{a_j\}_{j=1}^{N_A \le k_B} \leftarrow g_\phi^{\texttt{propose}}(o_{\text{curr}}, \mathcal{H}_{\text{curr}}, \mathcal{M}_{\text{ep}})$;
  if $\{a_j\}$ *is empty* **then**
    return a default action (e.g., random or no-op from available actions);
  $Q_{\text{root}}(a_j) \leftarrow -\infty$ for all a_j;
  foreach *candidate action* $a_j \in \{a_j\}$ **do**
    $(o'_j, r_j, d'_j) \leftarrow g_\phi^{\texttt{simulate}}(o_{\text{curr}}, a_j, \mathcal{H}_{\text{curr}}, \mathcal{M}_{\text{ep}})$;
    if d'_j **then**
      $V(z'_j) \leftarrow 0$; `# Terminal state, no future rewards`
    else
      $\mathcal{H}'_j \leftarrow \mathcal{H}_{\text{curr}} \oplus (\texttt{FormatAsHistoryString}(\texttt{"Act:"}, a_j), \texttt{FormatAsHistoryString}(\texttt{"Obs:"}, o'_j))$;
      $V(z'_j) \leftarrow \texttt{EstimateNodeValue}(o'_j, \mathcal{H}'_j, \mathcal{M}_{\text{ep}}, D_s - 1, k_B, \gamma, \lambda_{\text{step}}, g_\phi)$;
    $Q_{\text{root}}(a_j) \leftarrow r_j - \lambda_{\text{step}} + \gamma \cdot V(z'_j)$;
  return $\arg\max_{a_j} Q_{\text{root}}(a_j)$;

Function `EstimateNodeValue`$(o_{node}, \mathcal{H}_{node}, \mathcal{M}_{ep}, depth, k_B, \gamma, \lambda_{step}, g_\phi)$**:**
  if $depth \le 0$ ***or*** o_{node} *is known/simulated as terminal* **then**
    return $g_\phi^{\texttt{value}}(o_{\text{node}}, \mathcal{H}_{\text{node}}, \mathcal{M}_{\text{ep}})$;
  Candidate actions $\{a_k\} \leftarrow g_\phi^{\texttt{propose}}(o_{\text{node}}, \mathcal{H}_{\text{node}}, \mathcal{M}_{\text{ep}})$;
  if $\{a_k\}$ *is empty* **then**
    return $g_\phi^{\texttt{value}}(o_{\text{node}}, \mathcal{H}_{\text{node}}, \mathcal{M}_{\text{ep}})$; `# Leaf node: estimate value directly`
  $V_{\text{node\_val}} \leftarrow -\infty$; `# This will store` $\max_k Q(z_{\texttt{node}}, a_k)$
  foreach *action* $a_k \in \{a_k\}$ **do**
    $(o'_k, r_k, d'_k) \leftarrow g_\phi^{\texttt{simulate}}(o_{\text{node}}, a_k, \mathcal{H}_{\text{node}}, \mathcal{M}_{\text{ep}})$;
    if d'_k **then**
      $V(z'_k) \leftarrow 0$;
    else
      $\mathcal{H}'_k \leftarrow \mathcal{H}_{\text{node}} \oplus (\texttt{FormatAsHistoryString}(\texttt{"Act:"}, a_k), \texttt{FormatAsHistoryString}(\texttt{"Obs:"}, o'_k))$;
      $V(z'_k) \leftarrow \texttt{EstimateNodeValue}(o'_k, \mathcal{H}'_k, \mathcal{M}_{\text{ep}}, \text{depth} - 1, k_B, \gamma, \lambda_{\text{step}}, g_\phi)$;
    $Q(z_{\text{node}}, a_k) \leftarrow r_k - \lambda_{\text{step}} + \gamma \cdot V(z'_k)$;
    $V_{\text{node\_val}} \leftarrow \max(V_{\text{node\_val}}, Q(z_{\text{node}}, a_k))$;
  return $V_{\text{node\_val}}$; `# Node's value is max Q of children`

Recursive Lookahead Plan (Algorithm 2) This algorithm describes the planning process to select an action at the current step t.

- **Function** `RecursiveLookaheadPlan` **(Lines 1-12):** This is the entry point for planning at the root of the search (current actual state).

  - **Inputs:** Current observation o_{curr}, current short-term history $\mathcal{H}_{\text{curr}}$, the episode's Fact Memory $\mathcal{M}_{\text{ep}}$, max search depth D_s, branch factor k_B, discount γ, step penalty λ_{step}, and the planner LLM collection g_ϕ.
  - **Root Action Proposal (Line 2):** g_ϕ^{propose} generates initial candidate actions $\{a_j\}$ from o_{curr}. If no actions are proposed, a default policy is invoked (Lines 3-4).
  - **Root Q-Value Calculation (Lines 6-13):** For each proposed root action a_j:
    * The world model g_ϕ^{simulate} predicts the next state o'_j, immediate reward r_j, and done status d'_j.
    * If the simulated state o'_j is terminal (d'_j is true), its future value $V(z'_j)$ is 0 (Line 8).
    * Otherwise, the value $V(z'_j)$ is obtained by calling `EstimateNodeValue` (Line 10) for state o'_j with remaining depth $D_s - 1$. The history $\mathcal{H}'_j$ for this recursive call is the current history $\mathcal{H}_{\text{curr}}$ extended by the action a_j and simulated observation o'_j.
    * The Q-value $Q_{\text{root}}(a_j)$ is computed using the simulated reward r_j, the step penalty λ_{step}, and the discounted estimated value $V(z'_j)$ of the next state.
  - **Action Selection (Line 14):** The action a_t^* with the highest Q_{root} value is selected.

- **Function `EstimateNodeValue` (Lines 16-31):** This function recursively estimates the value of a node o_{node} in the search tree.
  - **Inputs:** The node's observation o_{node} and its history path $\mathcal{H}_{\text{node}}$, $\mathcal{M}_{\text{ep}}$, current remaining search depth `depth`, and other parameters.
  - **Base Cases (Lines 16-20):**
    * If `depth` ≤ 0, or if o_{node} is determined to be a terminal state, the recursion stops. The value of o_{node} is then directly estimated by g_ϕ^{value}.
    * If g_ϕ^{propose} fails to generate any actions from o_{node}, o_{node} is also treated as a leaf, and its value is estimated by g_ϕ^{value}.
  - **Recursive Step (Lines 22-30):** If not a base case:
    * Candidate actions $\{a_k\}$ are proposed from o_{node} using g_ϕ^{propose}.
    * For each action a_k: g_ϕ^{simulate} yields (o'_k, r_k, d'_k).
    * If d'_k is true, $V(z'_k) = 0$. Else, `EstimateNodeValue` is called recursively for o'_k with `depth-1` to get $V(z'_k)$.
    * The Q-value $Q(z_{\text{node}}, a_k)$ is calculated: $r_k - \lambda_{\text{step}} + \gamma V(z'_k)$.
  - **Return Value (Line 31):** The function returns $V_{\text{node\_val}} = \max_{a_k} Q(z_{\text{node}}, a_k)$, representing $V(z_{\text{node}})$.

  All LLM calls by g_ϕ components within the planning phase operate with a temperature of 0.0 for deterministic evaluations. Results are memoized within a single planning step (`_value_cache` in the implementation) to avoid redundant computations, as mentioned in Section 3.

Algorithm 3: LWM-Planner: Fact Model Learning and Memory Update

```
1 Function LearnFactsAndUpdateMemory(B_e, M_known, env_desc_str, Ψ_LLM)
    # Fact Extraction from completed episode trajectory
2   trajectory_summary_str ← FormatTrajectorySummary(B_e);
3   ΔF_e ← Ψ_LLM^extract(trajectory_summary_str, M_known, env_desc_str) ;      # Invokes fact elicitation LLM
    # Update Fact Memory and Optionally Refine/Compress
4   M_candidate ← M_known ∪ ΔF_e;
5   if compression hyperparameter is enabled then
6     M_next_global ← Ψ_LLM^refine(M_candidate, env_desc_str) ;                # Invokes fact compression LLM
7   else
8     M_next_global ← M_candidate;
9   return M_next_global;
```

Fact Model Learning and Memory Update (Algorithm 3) This procedure, corresponding to the `reflect` method in the agent's codebase, is executed at the end of each episode e to update the agent's knowledge.

- **Inputs (Line 1):** The episode trajectory buffer $\mathcal{B}_e$, the Fact Memory $\mathcal{M}_{\text{known}}$ that was used for planning during that episode, a description of the environment `env_desc_str`, and the Fact Extractor & Refiner LLM Ψ_{LLM}.
- **Fact Extraction (Lines 2-3):** A textual summary (`trajectory_summary_str`) of the trajectory in $\mathcal{B}_e$ is created. The fact elicitation component, $\Psi_{\text{LLM}}^{\text{extract}}$, processes this summary, along with $\mathcal{M}_{\text{known}}$ and `env_desc_str`, to generate a set of new candidate atomic facts $\Delta\mathcal{F}_e$.
- **Memory Update and Refinement (Lines 4-8):** The newly extracted facts $\Delta\mathcal{F}_e$ are combined with $\mathcal{M}_{\text{known}}$. If fact compression is enabled (via the `compress` flag in the implementation), $\Psi_{\text{LLM}}^{\text{refine}}$ processes this combined set to produce the refined Fact Memory $\mathcal{M}_{\text{next\_global}}$. Otherwise, the combined set becomes $\mathcal{M}_{\text{next\_global}}$.
- **Output (Line 9):** Returns the updated global Fact Memory $\mathcal{M}_{\text{next\_global}}$.

This iterative cycle enables LWM-Planner to adapt by progressively building a more accurate symbolic understanding of its environment.

D.1 Reference Implementation Notes

Algorithms 1–3 specify the high-level logic, while Appendices B, C, and N.2 specify the prompt structure and truncation policy. The present subsection fixes the concrete runtime contract needed for a faithful reimplementation of the core method. It is intended as a paper-native reference specification: an external engineer or LLM agent with access to a function-calling LLM and a text environment should be able to reproduce the core LWM-Planner loop from the description below.

Minimal environment interface. A reimplementation only requires the following environment API:

1. `reset() -> observation_string`, returning the initial textual observation for a fresh episode.
2. `step(action_string) -> (next_observation_string, reward_float, done_bool, info_dict)`.
3. A textual `env_description` string shared across all methods.
4. A finite text action set `action_space`, exposed as an ordered list or tuple of legal action strings.

No simulator internals, latent state access, or handcrafted symbolic parser are required.

Structured LLM-call interface. The core method can be implemented using five structured calls:

1. `ExtractFacts(trajectory_summary, known_facts, env_description) -> list[str]`
2. `CompressFacts(candidate_facts, env_description) -> list[str]`
3. `ProposeActions(observation, history, facts, env_description, allowed_actions, k_B) -> list[str]`
4. `SimulateStep(observation, action, history, facts, env_description) -> (next_observation, reward, done)`
5. `EstimateValue(observation, history, facts, env_description, gamma) -> float`

For reproducibility, canonicalize returned action strings and fact strings to lowercase, deduplicate exact string matches while preserving order, and truncate proposed actions to at most k_B unique actions.

Reference execution protocol. The following ordering matches the core method used throughout the paper:

1. **Episode reset.** Call `env.reset()`, clear the episode buffer, clear the short-term history, append the initial observation string, and snapshot the current global Fact Memory for the whole episode.

Table 3: **Persistent runtime state for a faithful reimplementation of LWM-Planner.** The key point is that facts are updated only between episodes, while planning state is local to a single decision.

State	Type	Role and update rule
Short-term history $\mathcal{H}$	deque of strings	Stores recent interaction strings in chronological order. At episode start, clear it and append the initial observation as `Obs: ...`. During real interaction, append `Act: ...` then `Obs: ...`. During search, extend local copies of this same template along simulated branches.
Global Fact Memory $\mathcal{M}$	ordered deque/list of strings	Persistent atomic fact memory across episodes. Snapshot it once at episode start and keep that snapshot fixed for all planning within the episode. Update the global memory only after the episode ends.
Episode buffer $\mathcal{B}_e$	list of tuples	Stores the realized episode trajectory as $(o_t, a_t, r_t, o_{t+1}, d_{t+1})$. This buffer is consumed by the fact-extraction stage at episode end, then cleared on the next reset.
Per-decision cache $\mathcal{C}$	dictionary	Memoizes `propose`, `simulate`, and `value` calls within one real action selection. Clear it before every new real decision. A faithful key is `(call_type, observation, action if any, history_tuple)`.
Terminal set $\mathcal{T}$	set of strings	Optional cache of observations already classified as terminal by the simulator. If a simulated observation is in $\mathcal{T}$, return future value 0 without further expansion.

2. **Planning state reset.** Before each real action selection, clear the per-decision cache. Keep the episode-level fact snapshot fixed; do not update facts mid-episode.
3. **Action proposal.** Query `ProposeActions` on the current observation, recent history, episode fact snapshot, environment description, and legal action set. Deduplicate exact action matches and keep the first k_B unique actions returned by the model.
4. **Single-step simulation.** For each proposed action, call `SimulateStep`. Extend the local branch history with `Act: ...` followed by the predicted `Obs: ...`. If the simulator predicts `done=True`, define the future value of that child to be 0.
5. **Leaf valuation.** If the remaining search depth is zero, or if an internal node produces no candidate actions, call `EstimateValue` on that node rather than expanding further.
6. **Recursive backup.** Compute action values using

$$Q(o_t, a_i) = r' - \lambda_{\text{step}} + \gamma \hat{V}(o').$$

   At each internal node, return the maximum child Q-value. At the root, choose the first action among the argmax set to keep planning deterministic.
7. **Real environment transition.** Execute the chosen action in the real environment, append $(o_t, a_t, r_t, o_{t+1}, d_{t+1})$ to the episode buffer, then append the realized `Act: ...` and `Obs: ...` strings to the real history deque.
8. **End-of-episode fact update.** When the episode terminates, construct a trajectory summary containing the outcome, total reward, and formatted step-by-step transition list. Call `ExtractFacts` with this summary, the previous Fact Memory, and the environment description. Merge the new facts with the old memory; if compression is enabled, run `CompressFacts` once on the merged set. The resulting fact memory is then used only from the next episode onward.

If `ProposeActions` returns no action at the root, a robust reimplementation should execute a fixed legal fallback action (for example, the first legal action in the environment's ordering) rather than leaving the behavior undefined. In our reported runs this failure mode was rare, but the guard is useful for portability.

Evaluation defaults. Unless otherwise stated, each run uses a fixed interaction budget of 300 environment steps and reports cumulative return over that budget. Results are averaged over three random seeds in most

Table 4: **Concrete defaults for the reported runs.** These values pin down the implementation choices that most strongly affect paper-faithful reproduction. Smaller search depths appear only in ablations and appendix-only robustness experiments.

Method	Core defaults	Notes
ReAct	History cap H_L=51 items; ReAct thought/action temperature 0.3; max output tokens 8512	No persistent memory beyond the short-term history buffer. Uses the shared environment header and action set.
Reflexion	ReAct defaults; lesson buffer length 5; post-episode lesson synthesis	Lessons are short natural-language summaries appended between episodes. No lookahead search.
ReAct + FEC	ReAct defaults; fact buffer length 200; fact extraction and compression temperature 0.0; compression enabled	Uses the same atomic-fact memory as LWM-Planner, but acts greedily without tree search.
LWM-Planner	Fact buffer length 200; planning depth d=3 in the main tables; branch factor k_B=4; planning discount γ=0.99; step penalty magnitude λ_{step}=0.02; proposal/simulation/value temperatures 0.0; fact extraction/compression temperature 0.0; max output tokens 8512	Facts are frozen within an episode and refreshed only between episodes. Planning calls share the same environment header and use local branch histories with per-decision memoization.

experiments due to inference-time cost; the main comparison table uses ten seeds where indicated in the caption. Appendix P specifies the remaining prompt-parity constraints shared across methods.

E Theory

This section provides the theoretical underpinnings of our fact-based reinforcement learning approach, focusing on the performance guarantees of an idealized agent.

E.1 Theorem: Performance of Idealized Fact-Based Agent (IFBA) with Perfect Abstract Model

Let π_F be the policy derived by the Idealized Fact-Based Agent (IFBA), as defined in Section 2.2 of the main text. If the ideal fact abstraction Ψ^* establishes an ϵ_{sim}-approximate bisimulation (Definition 2.2) between the ground MDP $\mathcal{G}$ and the induced abstract MDP M_{Ψ^*}, and the planner for M_{Ψ^*} is ϵ_{plan}-optimal (Definition 2.3), then for any state $s \in \mathcal{S}$:

$$V^*_{\mathcal{G}}(s) - V^{\pi_F}_{\mathcal{G}}(s) \leq \frac{2\epsilon_{\text{sim}}}{1-\gamma} + \epsilon_{\text{plan}} \tag{7}$$

E.1.1 Proof

The value loss of the policy π_F in the ground MDP $\mathcal{G}$ can be decomposed. Recall that $\pi_F(s') = \pi^\circ_{M_{\Psi^*}}(\Psi^*(s'))$, where $\pi^\circ_{M_{\Psi^*}}$ is the ϵ_{plan}-optimal policy found by the planner operating in the abstract MDP M_{Ψ^*}.

We express the total value loss as follows:

$$\begin{aligned} V^*_{\mathcal{G}}(s) - V^{\pi_F}_{\mathcal{G}}(s) = &\left(V^*_{\mathcal{G}}(s) - V^*_{M_{\Psi^*}}(\Psi^*(s))\right) \\ &+ \left(V^*_{M_{\Psi^*}}(\Psi^*(s)) - V^{\pi^\circ_{M_{\Psi^*}}}_{M_{\Psi^*}}(\Psi^*(s))\right) \\ &+ \left(V^{\pi^\circ_{M_{\Psi^*}}}_{M_{\Psi^*}}(\Psi^*(s)) - V^{\pi_F}_{\mathcal{G}}(s)\right) \end{aligned}$$

Let's analyze each term:

1. **Term I: Abstraction Error on the Optimal Value Function**
   This term, $\left(V^*_{\mathcal{G}}(s) - V^*_{M_{\Psi^*}}(\Psi^*(s))\right)$, represents the difference between the optimal value function in the ground MDP and the optimal value function in the abstract MDP, mapped back to the ground state via Ψ^*. By Definition 2.2 (specifically, Equation (1) in the main text), the property of ϵ_{sim}-approximate bisimulation directly bounds this difference:
   $$|V^*_{\mathcal{G}}(s) - V^*_{M_{\Psi^*}}(\Psi^*(s))| \leq \frac{\epsilon_{\text{sim}}}{1-\gamma}$$

2. **Term II: Planning Error in the Abstract MDP**
   This term, $\left(V^*_{M_{\Psi^*}}(\Psi^*(s)) - V^{\pi^\circ_{M_{\Psi^*}}}_{M_{\Psi^*}}(\Psi^*(s))\right)$, quantifies the sub-optimality of the policy $\pi^\circ_{M_{\Psi^*}}$ computed by the planner within the abstract model M_{Ψ^*}. By Definition 2.3, the planner is ϵ_{plan}-optimal, which means:
   $$V^*_{M_{\Psi^*}}(\Psi^*(s)) - V^{\pi^\circ_{M_{\Psi^*}}}_{M_{\Psi^*}}(\Psi^*(s)) \leq \epsilon_{\text{plan}}$$
   Since value functions are non-negative (assuming non-negative rewards or appropriate initialization), this term is bounded by ϵ_{plan}.

3. **Term III: Abstraction Error on the Planned Policy's Value**
   This term, $\left(V^{\pi^\circ_{M_{\Psi^*}}}_{M_{\Psi^*}}(\Psi^*(s)) - V^{\pi_F}_{\mathcal{G}}(s)\right)$, captures the difference between the value of the planned policy $\pi^\circ_{M_{\Psi^*}}$ when evaluated in the abstract MDP M_{Ψ^*} versus its value when executed in the ground MDP $\mathcal{G}$ (which is $V^{\pi_F}_{\mathcal{G}}(s)$ by definition of π_F). A key property of ϵ_{sim}-approximate bisimulation is that it not only bounds the difference in optimal value functions but also the difference in value functions for any fixed policy that respects the abstraction Ψ^*. Since $\pi_F(s') = \pi^\circ_{M_{\Psi^*}}(\Psi^*(s'))$, it respects the abstraction. Therefore, we have (Ravindran and Barto, 2004):
   $$|V^{\pi^\circ_{M_{\Psi^*}}}_{M_{\Psi^*}}(\Psi^*(s)) - V^{\pi_F}_{\mathcal{G}}(s)| \leq \frac{\epsilon_{\text{sim}}}{1-\gamma}$$

Combining the Bounds:

Using the triangle inequality ($X - Z = (X - Y) + (Y - Z) \implies |X - Z| \leq |X - Y| + |Y - Z|$), we can sum the absolute bounds of these terms. More directly, since Term II is already an upper bound on the difference (not an absolute value), we have:

$$\begin{aligned} V^*_{\mathcal{G}}(s) - V^{\pi_F}_{\mathcal{G}}(s) &\leq \left|V^*_{\mathcal{G}}(s) - V^*_{M_{\Psi^*}}(\Psi^*(s))\right| \\ &\quad + \left(V^*_{M_{\Psi^*}}(\Psi^*(s)) - V^{\pi^\circ_{M_{\Psi^*}}}_{M_{\Psi^*}}(\Psi^*(s))\right) \\ &\quad + \left|V^{\pi^\circ_{M_{\Psi^*}}}_{M_{\Psi^*}}(\Psi^*(s)) - V^{\pi_F}_{\mathcal{G}}(s)\right| \\ &\leq \frac{\epsilon_{\text{sim}}}{1-\gamma} + \epsilon_{\text{plan}} + \frac{\epsilon_{\text{sim}}}{1-\gamma} \\ &= \frac{2\epsilon_{\text{sim}}}{1-\gamma} + \epsilon_{\text{plan}} \end{aligned}$$

This completes the proof, establishing the performance bound for the Idealized Fact-Based Agent operating with a perfect abstract model M_{Ψ^*} derived from an ϵ_{sim}-approximate bisimulation Ψ^*, and an ϵ_{plan}-optimal planner. □

E.2 Further Theoretical Considerations

The theoretical framework presented in Section 2 and the proof in Appendix E.1.1 rely on the quality of the fact-based abstraction Ψ and the learned abstract model $\tilde{M}_\Psi$. Here, we elaborate on two guiding principles relevant to achieving a good abstraction and, consequently, robust agent performance: bisimulation for state aggregation and the Information Bottleneck principle for fact minimality and relevance.

E.2.1 Approximate State Abstraction and Bisimulation

The concept of ϵ_{sim}-approximate bisimulation (Definition 2.2) provides a formal grounding for why a good fact-based abstraction can lead to provably near-optimal policies (Ferns et al., 2004). A bisimulation groups states of the ground MDP $\mathcal{G}$ that are *behaviorally equivalent*. In an exact bisimulation, states s_1, s_2 are in the same abstract state $z = \Psi(s_1) = \Psi(s_2)$ if, for any action $a \in \mathcal{A}$:

- They yield the same immediate reward: $R(s_1, a) = R(s_2, a)$.
- They transition to the same distribution over next abstract states: $\sum_{s'_1 \in \mathcal{S}_{z'}} \mathcal{T}(s'_1|s_1, a) = \sum_{s'_2 \in \mathcal{S}_{z'}} \mathcal{T}(s'_2|s_2, a)$ for all $z' \in \mathcal{Z}_{\mathcal{F}}$.

The ϵ_{sim} term in an approximate bisimulation relaxes these conditions, allowing for small differences in one-step rewards and transition probabilities for states mapped to the same abstract state z. The crucial implication, as shown in Equation (1), is that the value function $V^*_{M_{\Psi^*}}$ in the abstract MDP M_{Ψ^*} (induced by an ϵ_{sim}-approximate bisimulation Ψ^*) is close to the optimal value function $V^*_{\mathcal{G}}$ in the ground MDP, with the error bounded by $\frac{\epsilon_{\text{sim}}}{1-\gamma}$.

In LWM-Planner, the atomic fact set F_s generated by the Fact Extractor f_θ for a ground state s forms the basis of the abstract state $z = \Psi(s) = (o, F_s)$ (where o is the direct observation). The goal of learning "critical" and "minimal" facts is to construct an abstraction Ψ that minimizes ϵ_{sim}. That is, the facts should capture enough information to ensure that states s_1, s_2 mapped to the same (o, F_s) have similar optimal values and similar optimal actions. If the extracted facts fail to distinguish between ground states that are behaviorally different (e.g., one leads to a high reward and another to a low reward with the same action), ϵ_{sim} will be large, and the performance guarantee in Equation (6) degrades. The online learning of facts is a process of refining Ψ to better approximate a bisimulation relevant to the task.

E.2.2 Information Bottleneck Principle and Fact Relevance

The desire for "minimal, yet impactful, units of knowledge" (atomic facts) aligns with the **Information Bottleneck (IB)** principle (Tishby et al., 2000; Alemi et al., 2017). The IB principle seeks to find a compressed representation

(bottleneck) Z of a source variable X that is maximally informative about a target relevance variable Y, while minimizing the mutual information $I(X;Z)$ (i.e., Z should be a minimal sufficient statistic of X for predicting Y).

In our context:

- The source variable X is the ground state s_t (or the full trajectory history leading to s_t).
- The compressed representation Z is the abstract state $z_t = \Psi(s_t)$, primarily defined by the set of atomic facts F_{s_t} derived from s_t or its history.
- The target relevance variable Y can be conceptualized as the optimal action-value function $Q^*_{\mathcal{G}}(s_t, a)$, the optimal value function $V^*_{\mathcal{G}}(s_t)$, or even future rewards and observations.

The LWM-Planner's fact extraction process aims to learn a Ψ (instantiated by f_θ) that distills F_{s_t} such that it is highly informative about $V^*_{\mathcal{G}}(s_t)$ (to minimize ϵ_{sim}) and future transitions/rewards (to minimize δ_{model} when g_ϕ uses these facts). The "atomicity" and "minimality" criteria for facts directly serve the goal of compressing s_t into F_{s_t} without losing task-critical information. A smaller, more relevant set of facts:

- **Reduces computational burden**: Shorter prompts for the LLM components.
- **Focuses LLM reasoning**: Prevents dilution of critical information within the LLM's limited context window, allowing the LLM to better utilize its pre-trained knowledge by grounding it on the most salient task-specific evidence.
- **Improves sample efficiency for model learning**: A smaller abstract state space $|\mathcal{Z}_{\mathcal{F}}|$ generally means that learning the dynamics $\tilde{\mathcal{T}}_\Psi$ and rewards $\tilde{R}_\Psi$ of the abstract MDP $\tilde{M}_\Psi$ (implicitly done by g_ϕ) requires fewer samples (interactions) to achieve a low δ_{model} (Strehl et al., 2009).

The optional fact compression step in LWM-Planner explicitly attempts to enforce this minimality by removing redundant or less informative facts, further aligning with the IB objective of finding a maximally compressed yet sufficient representation. The quality of this compression directly impacts the complexity and effectiveness of the learned abstract model and, consequently, the overall planning performance.

By striving for fact-based abstractions that approximate bisimulations and adhere to information bottleneck principles, LWM-Planner aims to construct an internal representation that is both robust for planning and efficient for learning. The interplay between the quality of facts (ϵ_{sim}) and the LLM's ability to simulate and value using these facts (δ_{model}) is central to achieving near-optimal performance, as formalized in Equation (6).

E.2.3 Future Theoretical Directions and Open Questions

While the current theoretical framework provides initial motivation, several avenues for future theoretical work could further solidify and extend the understanding of LWM-Planner and similar fact-based LLM agents.

Causality in Fact Extraction A significant direction is the integration of **causal reasoning** into the fact extraction process (Pearl, 2009). Currently, atomic facts are evaluated in terms of correlation[1], capturing observed relationships (e.g., "action Z leads_to_failure_condition"). However, facts that represent causal relationships (e.g., "action Z *causes* failure condition *if* precondition P holds") would offer more robust and generalizable knowledge.

- **Improved Generalization**: Causal facts are more likely to hold true under slight variations in the environment or task, potentially leading to a smaller δ_{model} when the agent encounters novel situations that share underlying causal structures.
- **Intervention-based Learning**: Future agents could be designed to perform specific interventions (exploratory actions) aimed at discovering causal links, rather than passively observing correlations. This could lead to more sample-efficient learning of highly impactful facts.

[1] LLMs probably implicitly already perform some form of causal reasoning

- **Counterfactual Reasoning**: A fact base enriched with causal understanding could allow the Planner LLM (g_ϕ) to engage in more sophisticated counterfactual reasoning during lookahead search (e.g., "what would have happened if I had chosen action B instead of A, given these causal rules?").

Developing methods for f_θ to reliably infer or validate causal statements from observational and interventional data within interactive environments is a challenging but promising research area.

Formal Analysis of LLM-driven Components The current framework treats the LLM components (g_ϕ^{propose}, g_ϕ^{simulate}, g_ϕ^{value}, and Ψ_{LLM}) largely as oracles with certain performance characteristics (e.g., LLM simulation accuracy contributing to δ_{model}). Future work could delve into:

- **Characterizing LLM Errors**: More formally characterizing the types of errors LLMs make in simulation and value estimation, and how these errors propagate through the lookahead search to affect ϵ_{plan}.
- **Impact of Prompt Engineering**: Theoretically analyzing the sensitivity of ϵ_{sim} and δ_{model} to the quality and structure of prompts, including the presentation of atomic facts.
- **Convergence of Fact Memory**: Investigating conditions under which the learned Fact Memory $\mathcal{M}_t$ converges to a "sufficient" or "minimal" set of facts that guarantees a certain level of performance (e.g., bounding ϵ_{sim} below a desired threshold).

Fact-Based Abstractions in Partially Observable MDPs (POMDPs) The current theoretical analysis assumes an underlying MDP where states s_t are fully observable or can be derived into a sufficient structured representation. Many real-world scenarios are better modeled as POMDPs (Kaelbling et al., 1998), where the agent receives observations o_t that are incomplete or noisy manifestations of the true underlying state s_t.

- **Belief State Abstraction**: Future work could explore how atomic facts can be used to form abstractions not directly over s_t, but over belief states $b(s_t) = P(s_t|\text{history})$. Atomic facts could represent properties of the environment that are inferred to be true with high probability based on the observation history.
- **Information-Gathering Facts**: The agent might learn facts not just about the environment's dynamics, but about which actions are most informative for reducing uncertainty about critical, unobserved aspects of the state, guiding exploration more effectively.

Addressing these theoretical questions will be crucial for advancing the capabilities and understanding of LLM agents that learn and plan from interaction by building and reasoning over symbolic knowledge representations like atomic facts.

F Benchmark Environments Details

We evaluate our LWM-Planner agent and baseline methods on three distinct, procedurally generated, text-based environments. Each environment is designed to test different aspects of an agent's learning and planning capabilities, ranging from grid-world navigation with sparse rewards to complex instruction following and multi-step crafting tasks.

F.1 TextFrozenLake

This environment is a procedurally generated text-based version of the classic FrozenLake problem (Brockman et al., 2016).

- **Objective**: The agent must navigate from a starting position (S) at coordinates (0,0) to a goal position (G) at $(N-1, N-1)$ on an $N \times N$ grid. The grid also contains ice surfaces (.) and holes (H). Reaching the goal yields a reward of +1.0, falling into a hole yields -1.0, and all other steps yield 0.0. An episode terminates upon reaching the goal, falling into a hole, or exceeding a maximum step limit.
- **Procedural Generation**:
  - The grid size N (e.g., 4×4, 6×6, 8×8) and hole density h are configurable parameters at initialization.
  - A key feature is the guarantee of at least one solvable path from start to goal. This is achieved by first constructing a "Manhattan corridor" or a zig-zag safe path near the diagonal to connect (0,0) and $(N-1, N-1)$. Holes are then sampled on the remaining cells based on the `hole_density` parameter.
  - The environment can be seeded for deterministic board generation and agent starting position.
- **Observation Space**: The agent receives a textual observation describing its current state, e.g., "You are at (0, 1) on ice.". This provides local information (current coordinates and terrain type of the square it stands on) without revealing the global map layout.
- **Action Space**: The agent has four discrete actions: "up", "down", "left", "right". Actions that would move the agent off the grid boundaries result in the agent remaining in its current position but on the edge cell.
- **Rewards**: As described, +1.0 for goal, -1.0 for a hole, 0.0 otherwise.
- **Episode Termination**: An episode ends if the agent reaches the goal (G), falls into a hole (H), or if the `_step_count` reaches `max_steps`. The `max_steps` is set to $8 \times (N-1)$, which is four times the optimal path length in an empty grid.
- `env_description`: The environment provides a detailed textual description string that includes the grid size, start/goal locations, reward structure, maximum steps, and hole density, explicitly stating that a path to the goal is guaranteed.

F.2 ALFWorld

We utilize the standard ALFWorld (Action Learning From World) benchmark (Shridhar et al., 2020b), which involves text-based agents performing tasks in simulated household environments based on the ALFRED dataset (Shridhar et al., 2020a). The `AlfWorldEnv` class in our codebase acts as a thin adapter around the official ALFWorld text environment.

- **Objective**: The agent is given a high-level natural language instruction (e.g., "put some spraybottle on toilet") and must navigate the environment, interact with objects and receptacles, and manipulate objects to satisfy the goal condition.
- **Environment Structure**: ALFWorld environments are simulated indoor scenes (kitchens, bedrooms, bathrooms) containing various receptacles (e.g., cabinets, drawers, countertops, sinks) and objects (e.g., spraybottle, bowl, desklamp). Objects can be picked up, placed in/on receptacles, and sometimes manipulated (e.g., heated, cleaned, sliced, though these more complex interactions are often simplified or yield generic feedback in some wrapper implementations).

- **Observation Space**: The agent receives rich textual observations describing its current location, visible objects and receptacles, its inventory, and feedback from its previous action. The initial observation also includes the specific task goal.
- **Action Space**: The environment defines a set of canonical action templates such as "look", "inventory", "go to (receptacle)", "open (receptacle)", "close (receptacle)", "take (object) from (receptacle)", "move (object) to (receptacle)", "examine (something)", "use (object)", etc..
- **Rewards**: A reward of +1.0 is typically given upon successful completion of the task goal. All other steps yield a reward of 0.0.
- **Episode Termination**: An episode ends if the agent successfully completes the task or if the maximum number of steps (`max_steps`, typically configured from a YAML file, e.g., `base_config.yaml`) is reached.
- **Task Variability**: ALFWorld offers a diverse set of tasks (identified by `task_id`) categorized into different types (e.g., pick & place, heat & place, clean & place). For our experiments, we randomly sample tasks from the "eval_out_of_distribution" split, as specified in the `AlfWorldEnv` constructor. Each `task_id` corresponds to a unique environment configuration and goal.
- `env_description`: The `AlfWorldEnv` provides a comprehensive `env_description` string that outlines the nature of the environment, receptacles, objects, task structure, reward system, maximum steps, the full list of admissible actions, and examples of interaction, along with advice for the agent.

For some ablation studies or simpler scenarios, a minimal version, `AlfMiniEnv`, is also available. It features a single, deterministically generated room with a fixed set of receptacle and object types (e.g., "drawer", "shelf", "vase", "keychain") and a canonical goal like "put some vase in safe 1". This version allows for more controlled experimentation by simplifying the state and action space while retaining the core object interaction mechanics. It also features deterministic resets to a blueprint state or a newly seeded state.

F.3 CrafterMini

CrafterMini is a procedurally generated, text-only, miniaturized version of the Crafter environment (Hafner, 2021), designed to test planning for resource gathering and multi-step crafting.

- **Objective**: The primary goal is to craft an "iron_pickaxe". This requires a sequence of sub-goals: collecting raw materials (wood, stone, iron) and crafting intermediate tools (wood_pickaxe, stone_pickaxe).
- **Environment Structure**: The world is a $N \times N$ grid (default 5×5) with a toroidal (wrap-around) topology. Each tile can be grass, tree, stone, iron, or water.
- **Procedural Generation**:
  - The grid size N and `max_steps` are configurable. The world is seeded for deterministic generation.
  - The grid is randomly populated with tiles, ensuring that at least one of each crucial resource (tree, stone, iron) is present, making the game solvable.
  - The `reset()` method can either restore an initial "blueprint" of the world or generate a new world if a new seed is provided.
- **Observation Space**: The observation is a textual string describing the agent's current tile type and coordinates, the terrain in the four cardinal directions, the agent's inventory (e.g., "wood=3, stone=1"), and a list of tools already crafted.
- **Action Space**: Actions are represented by integers with corresponding names:
  - 0-3: Movement (north, south, east, west).
  - 4: "collect" - Gathers a resource from the current tile if it's a resource tile (tree, stone, iron). Collecting turns the tile to grass.
  - 5: "craft_wood_pickaxe" (requires 3 wood).
  - 6: "craft_stone_pickaxe" (requires 1 wood, 3 stone).

  - 7: "craft_iron_pickaxe" (requires 1 stone_pickaxe, 3 iron).

  Crafting actions are only considered available if the recipe can be satisfied by the current inventory and already crafted tools (e.g., a stone_pickaxe is consumed to make an iron_pickaxe). The `RobustCrafterMiniEnv` variant can also parse textual synonyms for these actions.

- **Rewards**:
  - Each step incurs a -1 reward (step cost).
  - Crafting a wood_pickaxe gives +10 reward.
  - Crafting a stone_pickaxe gives +20 reward.
  - Crafting an iron_pickaxe gives +50 reward.
- **Episode Termination**: The episode ends immediately if an "iron_pickaxe" is successfully crafted or if the number of steps reaches `max_steps` (default $4 \times N^2$).
- `env_description`: A detailed textual description outlines the grid, observation format, action list with integer mappings, crafting recipes, rewards, and termination conditions.

These three environments provide a diverse set of challenges for evaluating the LWM-Planner's ability to learn from textual interactions, build effective world models through atomic facts, and plan over extended horizons.

G Benchmark Method Implementation Details

This section provides a detailed overview of the high-level logic for each benchmark method evaluated in our experiments. All LLM-based agents (LWM-Planner, ReAct, Reflexion, and ReAct + FEC) utilize a frozen LLM (model `gpt-4o`). LLM interactions are performed via API calls using a common internal utility that supports structured function calling; no LLM weights are updated during experiments. The default temperature for LLM calls in planning components (simulation, value estimation) and fact processing is 0.0. For ReAct-style thought generation, a temperature of 0.3 is typically used. The maximum token output for LLM responses is configured to 8512 tokens. A compact implementation-facing specification of the core runtime contract and reported defaults is given in Appendix D.1.

G.1 Random Agent

The **Random** agent serves as a basic non-learning baseline.

- **Policy**: At each step, the agent selects an action uniformly at random from the set of allowed actions provided for the specific environment.
- **State**: It maintains a short-term memory buffer of observation-action pairs for logging consistency, though this history does not influence its action selection.

G.2 ReAct Agent

The **ReAct** agent implements the Reason-Act prompting paradigm (Yao et al., 2023).

- **Core Mechanism**: The agent prompts an LLM to first generate a "Thought" (internal reasoning) and then an "Action" to take in the environment.
- **LLM Interaction**:
  - A prompt is constructed using a template specific to the ReAct style. This template incorporates a description of the environment, the current observation, the recent interaction history (formatted as a sequence of observations and actions), and the list of allowed actions.
  - The LLM is expected to provide its output in a structured format that distinguishes the thought process from the chosen action.
- **State**: The agent maintains a short-term memory buffer (a deque of observation-action strings) of a configurable length (e.g., 51 interactions). It also stores the most recent thought generated by the LLM.
- **Hyperparameters**: Key parameters include the length of the interaction history, LLM model, temperature, and maximum token limits.

G.3 Reflexion Agent

The **Reflexion** agent extends the ReAct agent by incorporating a self-reflection mechanism to learn from past experiences (Shinn et al., 2023).

- **Core Mechanism**: In addition to ReAct's thought-action cycle, after each episode, the Reflexion agent analyzes its trajectory to generate a textual "lesson".
- **Lesson Generation and Usage**:
  - Lessons are stored in a memory buffer (a deque) of a configurable maximum length (e.g., 5 lessons).
  - The ReAct prompt template is augmented to include these learned lessons, providing additional context for future decisions.
  - A post-episode reflection process involves constructing a summary of the completed trajectory (including observations, actions, rewards, and overall outcome) and prompting an LLM to generate a concise, actionable lesson (typically ≤ 20 words and prefixed accordingly).

- **State**: In addition to the ReAct state, it maintains the buffer of lessons, a record of the current episode's trajectory (observations, actions, rewards, next observations), and the cumulative reward for the current episode.

- **Hyperparameters**: Includes ReAct parameters plus the lesson buffer length and a reward threshold to determine episode success for reflection purposes (e.g., 0.99).

G.4 ReAct + FEC Agent (Ablation)

This agent is an ablation of our full LWM-Planner. It combines the ReAct decision-making process with the Fact Extraction and Compression (FEC) mechanism, but without lookahead search.

- **Fact Mechanism**:
  - Maintains a memory buffer (a deque) of atomic facts with a configurable maximum length (e.g., 200 facts).
  - The ReAct prompt template is augmented to include these atomic facts.
  - **Fact Extraction**: After each episode, a dedicated LLM-driven process analyzes the trajectory summary, environment description, and existing facts to identify minimal new atomic facts critical for improving predictions. The LLM is guided to output these new facts in a structured manner.
  - **Fact Compression**: If enabled, another LLM-driven process reviews the complete set of current facts (newly extracted plus existing) along with the environment description. It aims to produce a concise, refined set of facts by removing redundancies or information trivially inferable from the environment description, again using a structured output format.

- **Decision-Making**: Employs the standard ReAct thought-action cycle, but the LLM's reasoning is informed by the dynamically updated set of atomic facts included in its prompt.

- **Hyperparameters**: Includes ReAct parameters plus the fact buffer length and a flag to enable/disable fact compression.

G.5 LWM-Planner (Our Method)

The LWM-Planner is our proposed agent that integrates online atomic fact learning with a recursive lookahead search, where LLMs serve as key planning components.

- **Base Functionality**: It incorporates the same fact extraction and compression mechanisms (Fact Extractor Ψ_{LLM}) as the ReAct + FEC agent. These facts are stored in a dynamically updated memory buffer (a deque with a capacity of, e.g., 200 facts) and are used to augment the reasoning of all LLM components.

- **Core Planning Mechanism**: Instead of a direct ReAct step, LWM-Planner performs a depth-limited recursive lookahead search to select actions (Planner g_ϕ).
  - **Action Proposal ($g_\phi^{\textbf{propose}}$)**: At each node in the search, an LLM module proposes a set of plausible actions (up to a configurable branching factor, e.g., 4). This proposal is conditioned on the current (potentially simulated) observation, the history of interactions within the simulation, and the accumulated atomic facts.
  - **Latent World Model Simulation ($g_\phi^{\textbf{simulate}}$)**: For each proposed action, an LLM module predicts the next (latent) observation, immediate reward, and termination status. This simulation is conditioned on the current state, the action being simulated, the simulation history, and the atomic facts. This LLM interaction operates with a temperature of 0.0 for deterministic outcomes.
  - **Value Estimation ($g_\phi^{\textbf{value}}$)**: At the leaves of the search tree (determined by a configurable search depth, e.g., 3, or upon reaching a terminal state), an LLM module estimates the discounted future cumulative reward (value) from that state. This estimation is also conditioned on the state's observation, simulation history, and atomic facts, and uses a temperature of 0.0.

- **Q-Value Computation**: The Q-value for an action a_i from observation o_t is computed as $Q(o_t, a_i) = r' - \lambda_{\text{step}} + \gamma \hat{V}(o')$, where r' and o' are from the simulation, λ_{step} is a step penalty (e.g., -0.01), and γ is the discount factor (e.g., 0.99). $\hat{V}(o')$ is the estimated value of the next state, derived either from further recursion or direct estimation at a leaf node.
- **State and Caching**: The agent maintains a short-term interaction history (e.g., last 51 interactions) and the set of learned atomic facts. Within a single planning phase (for one action selection), results of LLM calls for action proposal, simulation, and value estimation are memoized to avoid redundant computations. A set of known terminal observations is also maintained across steps.
- **Hyperparameters**: Inherits fact-related parameters. Key planning parameters include search depth, branching factor, discount factor for planning, and step penalty. Asynchronous execution of LLM calls for different search branches is supported to improve computational efficiency. The MCTS-based extension of this agent introduces further parameters like the number of simulations and an exploration constant (UCT).

All LLM-based agents are initialized with a textual description of the environment and the set of allowed actions. Their prompts are dynamically constructed to include the current observation, relevant interaction history, and any learned knowledge (lessons or facts) appropriate for the specific agent architecture.

H Evaluation Details

The experimental evaluation of our proposed LWM-Planner and baseline methods follows a structured procedure to ensure fair comparison and robust assessment of performance. Key aspects of our evaluation protocol are detailed below:

- **Run Duration**: Each agent method is run for a total of 300 environment steps per evaluation trial, unless specified otherwise in a particular experiment. Within these 300 steps, an agent may complete multiple episodes depending on task complexity and its efficiency.
- **Metrics Tracked**: We focus on the following quantitative metrics to assess agent performance:
  - **Cumulative Return**: This is the primary metric and is defined as the sum of all rewards obtained by the agent over the entire 300-step run. It reflects the agent's overall ability to accumulate reward within a fixed interaction budget.
  - **Steps per Success**: For environments that have a clear binary success condition (e.g., reaching the goal tile in TextFrozenLake, or successfully completing the assigned task in ALFWorld), we record the number of environment steps taken within an episode to achieve that success. This metric is typically reported as an average over all successful episodes completed by the agent during its run. If an agent fails to achieve success in an episode or across the entire 300-step run, it may not contribute to this average, or its contribution might be noted as not applicable (e.g., marked as '–' in results tables).
- **Replication and Statistical Significance**: To account for inherent stochasticity in agent learning (if applicable) and environment generation (for procedurally generated tasks), results for each method on each environment are averaged over multiple independent runs, each initialized with a different random seed. The number of seeds is typically 3 or 10, as specified in the respective table captions in Section 5. We report the mean of the metrics and the 95% confidence interval (CI) to provide a measure of statistical significance and variability.
- **Final Success Rate**: For ALFWorld we additionally report the percentage of tasks solved at least once (FSR). This measure is complementary to cumulative return and enables direct comparison to prior work that reports success rates only.
- **Computational Cost**: We record the total number of tokens processed, the average latency per environment step, and an estimated monetary cost assuming the public pricing of the deployed API; detailed figures are provided in Appendix J.6.
- **Type of Compute Used**: All experiments were run on a single workstation equipped with an Intel Core i9 CPU and an NVIDIA RTX 3090 GPU (24 GB VRAM), together with hosted LLM API calls configured as detailed in Section H.

I Case Study: LWM-Planner on TextFrozenLake (4x4)

This case study details the learning process of the LWM-Planner agent in a procedurally generated 4×4 TextFrozenLake environment. The specific instance, corresponding to `grid_4_h_9_s_0` from our experiments (with a 90% chance any non-start/goal tile is a hole, and solvability ensured), features a high density of holes. Holes (H) terminate the episode with a negative reward. The agent's objective is to navigate from the starting position (S) at `(0,0)` to the goal (G) at `(3,3)`. Ice tiles (.) are safe to traverse.

The initial state of the grid is (S represents the Start/Agent):

```
S . H H
H . . H
H H . .
H H H G
```

The agent never observes this map, this is just for illustration. Instead the agent can only learn where the holes are by falling down each hole at least once, motivating the need for a persistent memory.

Initial Exploration and Learning from Failures (Episodes 0-3)

In its initial episodes, LWM-Planner explores the environment and primarily learns by encountering hazards. The atomic fact extraction mechanism is crucial during this phase for building a rudimentary map of dangers.

- **Episode 0:** The agent's first action is to move 'down' from `(0,0)`.
  - Trajectory Snippet: `Obs: You are at (0,0) on start. | Act: down | R: -1.0 | Next: You are at (1,0) on hole.`
  - Outcome: FAILURE.
  - **Fact Extracted:** `(1,0) is a hole.`
  - **Accumulated Facts (after Ep. 0):** `['(1,0) is a hole.']`

  This first fact immediately informs the agent about a critical environmental feature.

- **Episode 1:** Aware of the hole at `(1,0)`, the agent attempts a different path. It moves 'right' from `(0,0)` to `(0,1)`, then 'down' to `(1,1)`, and then 'down' again.
  - Trajectory Snippet: `...Obs: You are at (1,1) on ice. | Act: down | R: -1.0 | Next: You are at (2,1) on hole.`
  - Outcome: FAILURE.
  - **Fact Extracted:** `(2,1) is a hole.`
  - **Accumulated Facts (after Ep. 1):** `['(2,1) is a hole.', '(1,0) is a hole.']`

- **Episode 2:** The agent continues to explore. From `(0,1)`, it moves 'right'.
  - Trajectory Snippet: `...Obs: You are at (0,1) on ice. | Act: right | R: -1.0 | Next: You are at (0,2) on hole.`
  - Outcome: FAILURE.
  - **Fact Extracted:** `(0,2) is a hole.`
  - **Accumulated Facts (after Ep. 2):** `['(0,2) is a hole.', '(2,1) is a hole.', '(1,0) is a hole.']`

- **Episode 3:** Another failed attempt reveals another hole. From `(1,2)`, it moves 'right'.
  - Trajectory Snippet: `...Obs: You are at (1,2) on ice. | Act: right | R: -1.0 | Next: You are at (1,3) on hole.`
  - Outcome: FAILURE.
  - **Fact Extracted:** `(1,3) is a hole.`

  - **Accumulated Facts (after Ep. 3):** `['(1,3) is a hole.', '(0,2) is a hole.', '(2,1) is a hole.', '(1,0) is a hole.']`

After these initial four failures, the agent has learned the locations of four distinct holes. This knowledge is critical for subsequent planning using lookahead search, as these facts help the LLM-based simulator predict negative outcomes.

First Success and Learning the Safe Path (Episode 4)

Equipped with knowledge of several hazards, LWM-Planner's lookahead search can now better evaluate potential paths, biasing away from known holes.

- **Episode 4:** The agent successfully navigates to the goal.
  - Full Trajectory: `(0,0) -> right -> (0,1) -> down -> (1,1) -> right -> (1,2) -> down -> (2,2) -> right -> (2,3) -> down -> (3,3)`
  - Steps: 6 (Optimal for this grid)
  - Outcome: SUCCESS (Reward: +1.0)
  - **Facts Extracted:** A set of facts confirming the nature of the traversed safe tiles and the goal location:
    * `(0,1) is ice.`
    * `(1,1) is ice.`
    * `(1,2) is ice.`
    * `(2,2) is ice.`
    * `(2,3) is ice.`
    * `(3,3) is the goal.`
  - **Accumulated Facts (Snapshot after Ep. 4 includes):** `['(0,1) is ice.', '(1,1) is ice.', ..., '(3,3) is the goal.', '(1,3) is a hole.', '(0,2) is a hole.', '(2,1) is a hole.', '(1,0) is a hole.']`

This successful episode significantly expands the agent's knowledge base, not just with more hazards, but with positive confirmation of safe (ice) tiles and the goal's location. This richer set of facts allows the LLM-driven world model and value estimator to make more accurate predictions during lookahead.

Refinement and Consistent Optimal Performance (Episodes 5 onwards)

Even after the first success, the agent continues to refine its understanding of the environment.

- **Episode 5:** The agent explores an alternative move from a state on the previously successful path (`(2,2)`) and encounters another hole.
  - Trajectory Snippet: `...Obs: You are at (2,2) on ice. | Act: down | R: -1.0 | Next: You are at (3,2) on hole.`
  - Outcome: FAILURE.
  - **Fact Extracted:** `(3,2) is a hole.`
  - This further completes the agent's map of hazards, particularly those adjacent to the known safe path.
- **Episode 6:** The agent again reaches the goal in 6 steps, following the optimal path.
  - Outcome: SUCCESS.
  - **Facts Extracted:** `['(0,0) is the start.', '(0,0) is ice.']` (Identifying properties of the start tile based on the successful trajectory.)
- **Episode 7:** The agent achieves another 6-step success. The LLM's reflection process identifies additional facts based on the episode's context and existing knowledge.
  - Outcome: SUCCESS.

  - **New Facts Extracted Include:** `'(0,3) is a hole.'`, `'(3,0) is a hole.'`, `'(3,1) is a hole.'`. The LLM also re-identified `'(3,2) is a hole.'` (learned in Episode 5), possibly due to its relevance in the broader context of successful navigation.
  - **Accumulated Facts (after Ep. 7):**

```
['(0,3) is a hole.', '(3,0) is a hole.', '(3,1) is a hole.',
'(3,2) is a hole.', '(0,0) is the start.', '(0,0) is ice.',
'(0,1) is ice.', '(1,1) is ice.', '(1,2) is ice.',
'(2,2) is ice.', '(2,3) is ice.', '(3,3) is the goal.',
'(1,3) is a hole.', '(0,2) is a hole.',
'(2,1) is a hole.', '(1,0) is a hole.']
```

  At this stage, the agent has a fairly comprehensive map of the 4x4 grid, identifying most holes and the safe path.

- **Subsequent Episodes (e.g., Episodes 8-15 from trace):** The agent consistently solves the task by taking the optimal 6-step path. Fact extraction continues to refine its knowledge. For example, in Episode 11, the fact `'(2,0) is a hole.'` is added, correctly identifying one of the remaining unknown holes. Other extracted facts often reinforce existing knowledge (e.g., `'(0,1) is not a hole.'` in Episode 8, consistent with `'(0,1) is ice.'`). By Episode 12, the agent's fact list implies knowledge of all hole locations and the optimal path.

Comparison with ReAct and Reflexion Baselines

To contextualize LWM-Planner's performance, we compare its learning trajectory with ReAct and Reflexion agents on the same TextFrozenLake instance (4×4, $h = 0.9$, seed 0).

ReAct Agent: The ReAct agent, which relies on in-context reasoning based on the current observation and a short interaction history, struggled significantly in this environment. Over 150 timesteps (spanning 83 episodes in the provided trace), the ReAct agent failed to solve the task even once.

- **Behavior Pattern:** ReAct repeatedly fell into the same holes. For example, it fell into the hole at `(1,0)` (by moving 'down' from start) in Episode 0, and repeated this exact mistake in Episodes 3, 4, 5, 7, 10, 11, 13, 14, 15, 16, etc. Similarly, it frequently fell into the hole at `(2,1)` (e.g., Episodes 1, 2, 6, 8, 9, 12).
- **Lack of Persistent Memory:** This behavior demonstrates ReAct's core limitation in environments requiring persistent spatial memory beyond its immediate prompt context. Without a mechanism to explicitly record and recall that `"(1,0) is a hole"` across episodes, it re-discovers these hazards repeatedly. The short-term history provided in its prompt is insufficient for building a persistent map of the environment.

ReAct's performance highlights the challenge of pure in-context reasoning without a structured memory mechanism for accumulating task-critical knowledge like hazard locations.

Reflexion Agent: The Reflexion agent incorporates an episodic self-reflection mechanism, generating textual "lessons" from past failures and successes. This allows for a degree of learning across episodes.

- **Initial Learning:** Reflexion also initially failed, but its lessons attempted to capture insights.
  - Ep 0 Failure: (fell into `(1,0)`) → Lesson: "Avoid moving into holes by evaluating the safety of the next position before taking an action." (General advice)
  - Ep 1 Failure: (fell into `(0,2)`) → Lesson: "Avoid moving right from (0,1) on ice to prevent falling into the hole and losing reward." (More specific, state-action advice)
- **First Success:** Reflexion achieved its first success (optimal 6 steps) in Episode 5 (after 11 total environment steps, plus 4 prior failed episodes). This is notably slower than LWM-Planner, which succeeded in Episode 4 (after 4 prior failed episodes, totaling 10 failure steps + 6 success steps = 16 steps to first success, vs Reflexion's 4 failure episodes of 1+2+3+4 = 10 steps + 6 success steps = 16 steps to first success – wait, the

LWM-Planner trace shows 1+3+2+4 = 10 steps for failures, so LWM-Planner also took 16 steps to first success).

- **Nature of Lessons vs. Facts:** Reflexion's lessons are typically higher-level strategic advice or state-action rules (e.g., "Avoid moving down from (1,1) on ice..."). While helpful, these lessons are less granular and less directly usable for precise world model simulation compared to LWM-Planner's atomic facts (e.g., `"(2,1) is a hole."`). An atomic fact directly describes a property of the environment state, which is crucial for simulating outcomes.
- **Consistency and Repeated Errors:** Despite learning, Reflexion still exhibited some inconsistent behavior and repeated errors. For instance, after its first success in Episode 5, it failed in Episode 6 by falling into `(3,2)` (a new hole). In Episode 7, it repeated the mistake from Episode 1 by falling into `(0,2)`, even though a lesson about it was generated. This suggests that the general nature of lessons or the limited buffer for lessons might not always prevent re-encountering hazards if the specific context isn't perfectly matched by an active lesson. It did achieve further successes (e.g., Ep 10, 11, 16, 17, 18), showing progressive improvement, but its path to consistent optimal play was slower and less robust than LWM-Planner's.

Reflexion demonstrates learning through its self-generated advice, but the abstract nature of its lessons and potential for lesson forgetting (due to a limited buffer) can make it less efficient and robust than LWM-Planner's fact-based learning in this type of task.

Analysis of LWM-Planner's Advantage

This case study, when compared to ReAct and Reflexion, demonstrates LWM-Planner's ability to:

1. **Learn from Failures and Successes:** Initial interactions quickly identify critical hazards (holes), and successful trajectories confirm safe paths and the goal. Both types of experiences are distilled into atomic facts.
2. **Improve Planning via Fact Augmentation:** The accumulated atomic facts dynamically augment the prompts for the LLM components (proposer, simulator, value estimator). This grounding significantly improves the LLM's ability to:
   - Simulate transitions more accurately (reducing δ_{model} from our theoretical framework): Knowing `'(1,0) is a hole.'` means simulating 'down' from `(0,0)` will correctly predict a terminal state and negative reward. This precise knowledge is more effective than ReAct's lack of memory or Reflexion's more general "avoid holes" advice.
   - Estimate state values more effectively: States adjacent to known holes or leading towards known safe paths to the goal will have more accurate value estimates, guiding the lookahead search. LWM-Planner's value estimation is directly informed by a growing, precise map.
   - Propose better actions: The action proposer is less likely to suggest actions leading directly into known holes because the facts make these outcomes predictable during the lookahead.
3. **Achieve Consistent Optimal Behavior More Quickly:** By building a sufficiently accurate and granular fact-based abstraction (Ψ) of the environment, LWM-Planner converges to an optimal policy for this TextFrozenLake instance more rapidly and consistently than Reflexion, and vastly outperforms ReAct. The agent's performance, in terms of steps to goal, rapidly improves and stabilizes at the optimal 6 steps after a few initial exploratory episodes.
4. **Leverage In-Context Learning with Structured Knowledge:** All learning occurs via prompt augmentation with dynamically generated, structured atomic facts, without any LLM weight updates. This showcases the power of in-context learning when guided by distilled, experience-derived knowledge that is directly usable for building an internal model of the environment.

The conciseness and specificity of atomic facts (e.g., `'(1,0) is a hole.'`) provide verifiable information that is directly usable by the LLM during its lookahead search. This contrasts with ReAct's inability to form such a persistent representation and Reflexion's more general textual advice.

This progression from exploration and failure to consistent, optimal task completion highlights the effectiveness of combining online atomic fact augmentation with LLM-driven lookahead search for adaptive planning and decision-making, particularly when compared to methods with less structured or less persistent learning mechanisms.

J Additional Results

J.1 ALFWorld Full Results

In the main paper we evaluate on three ALFWorld environments, of ALFWorld-A, ALFWorld-B, ALFWorld-C which correspond to tasks 90, 3, and 5 respectively. To ensure exhaustive evaluation we compare against all the ALFWorld evaluation environments (Shinn et al., 2023). We tabulate these in Tables 5 and 7 to 9.

Table 5: Aggregated performance across all ALFWorld 134 eval environments (single-seed runs, 95% CIs). Higher cumulative return ↑ is better.

Method (metric)	ALFWorld Aggregate
LWM-Planner (Cum. return ↑)	**10.42±1.43**
ReAct + FEC (Cum. return ↑)	8.53±0.93
ReAct (Cum. return ↑)	5.00±0.09
Reflexion (Cum. return ↑)	4.36±0.10
Random (Cum. return ↑)	0.00±0.00

Table 6: Environment-normalised success-rate on the 134 ALFWorld evaluation tasks (single-seed runs, 95% confidence intervals). Higher ↑ is better.

Method (metric)	ALFWorld Aggregate
LWM-Planner (Norm. SR ↑)	**71.5±6.1**
ReAct + FEC (Norm. SR ↑)	68.1±5.8
ReAct (Norm. SR ↑)	46.9±4.1
Reflexion (Norm. SR ↑)	37.4±4.1
Random (Norm. SR ↑)	0.0±0.0

Table 7: Cumulative return per ALFWorld task (0–50). Higher ↑ is better. Single-seed runs (no CI).

Environment	LWM-Planner ↑	ReAct + FEC ↑	ReAct ↑	Reflexion ↑	Random ↑
ALFWORLD-0	**12.00**	5.00	4.00	4.00	0.00
ALFWORLD-1	2.00	**16.00**	6.00	4.00	0.00
ALFWORLD-2	**10.00**	5.00	4.00	4.00	0.00
ALFWORLD-3	**10.00**	2.00	5.00	4.00	0.00
ALFWORLD-4	**8.00**	**8.00**	5.00	4.00	0.00
ALFWORLD-5	**4.00**	0.00	**4.00**	**4.00**	0.00
ALFWORLD-6	12.00	**14.00**	6.00	4.00	0.00
ALFWORLD-7	4.00	**17.00**	5.00	5.00	0.00
ALFWORLD-8	0.00	**12.00**	5.00	5.00	0.00
ALFWORLD-9	**6.00**	5.00	5.00	5.00	0.00
ALFWORLD-10	**12.00**	8.00	5.00	5.00	0.00
ALFWORLD-11	**12.00**	4.00	5.00	4.00	0.00
ALFWORLD-12	0.00	**8.00**	5.00	4.00	0.00
ALFWORLD-13	**7.00**	**7.00**	5.00	4.00	0.00
ALFWORLD-14	0.00	**15.00**	5.00	5.00	0.00
ALFWORLD-15	1.00	**6.00**	5.00	5.00	0.00
ALFWORLD-16	**11.00**	2.00	5.00	–	0.00
ALFWORLD-17	2.00	**15.00**	5.00	4.00	0.00
ALFWORLD-18	**18.00**	9.00	5.00	4.00	0.00
ALFWORLD-19	6.00	**11.00**	5.00	5.00	0.00
ALFWORLD-20	8.00	**9.00**	5.00	4.00	0.00
ALFWORLD-21	0.00	3.00	**5.00**	–	0.00
ALFWORLD-22	**9.00**	3.00	4.00	5.00	0.00
ALFWORLD-23	8.00	**12.00**	5.00	5.00	0.00
ALFWORLD-24	4.00	4.00	**5.00**	4.00	0.00
ALFWORLD-25	7.00	**10.00**	5.00	5.00	0.00
ALFWORLD-26	9.00	**13.00**	5.00	4.00	0.00
ALFWORLD-27	**24.00**	8.00	6.00	4.00	0.00
ALFWORLD-28	**6.00**	3.00	5.00	4.00	0.00
ALFWORLD-29	**13.00**	11.00	5.00	5.00	0.00
ALFWORLD-30	3.00	1.00	**4.00**	**4.00**	0.00
ALFWORLD-31	8.00	**11.00**	5.00	5.00	0.00
ALFWORLD-32	**11.00**	7.00	6.00	4.00	0.00
ALFWORLD-33	1.00	**8.00**	4.00	4.00	0.00
ALFWORLD-34	1.00	**5.00**	**5.00**	**5.00**	0.00
ALFWORLD-35	**26.00**	3.00	6.00	4.00	0.00
ALFWORLD-36	5.00	**7.00**	6.00	4.00	0.00
ALFWORLD-37	**14.00**	3.00	5.00	4.00	0.00
ALFWORLD-38	**10.00**	6.00	5.00	4.00	0.00
ALFWORLD-39	**23.00**	3.00	5.00	5.00	0.00
ALFWORLD-40	2.00	**13.00**	5.00	4.00	0.00
ALFWORLD-41	2.00	**16.00**	5.00	4.00	0.00
ALFWORLD-42	**20.00**	12.00	4.00	4.00	0.00
ALFWORLD-43	5.00	4.00	**6.00**	4.00	0.00
ALFWORLD-44	**11.00**	10.00	5.00	5.00	0.00
ALFWORLD-45	**7.00**	6.00	5.00	5.00	0.00
ALFWORLD-46	1.00	**8.00**	5.00	5.00	0.00
ALFWORLD-47	4.00	**8.00**	5.00	5.00	0.00
ALFWORLD-48	**13.00**	4.00	5.00	4.00	0.00
ALFWORLD-49	0.00	2.00	**5.00**	**5.00**	0.00
ALFWORLD-50	3.00	**5.00**	**5.00**	3.00	0.00

Table 8: Cumulative return per ALFWorld task (51–100). Higher ↑ is better. Single-seed runs (no CI).

Environment	LWM-Planner ↑	ReAct + FEC ↑	ReAct ↑	Reflexion ↑	Random ↑
ALFWORLD-51	**12.00**	9.00	5.00	4.00	0.00
ALFWORLD-52	**28.00**	4.00	5.00	4.00	0.00
ALFWORLD-53	**13.00**	12.00	5.00	4.00	0.00
ALFWORLD-54	**5.00**	4.00	**5.00**	–	0.00
ALFWORLD-55	**19.00**	6.00	6.00	4.00	0.00
ALFWORLD-56	**10.00**	9.00	5.00	5.00	0.00
ALFWORLD-57	**17.00**	4.00	5.00	5.00	0.00
ALFWORLD-58	9.00	**13.00**	5.00	4.00	0.00
ALFWORLD-59	**26.00**	9.00	6.00	4.00	0.00
ALFWORLD-60	**29.00**	0.00	5.00	5.00	0.00
ALFWORLD-61	**23.00**	8.00	6.00	5.00	0.00
ALFWORLD-62	7.00	**8.00**	5.00	4.00	0.00
ALFWORLD-63	12.00	**21.00**	5.00	4.00	0.00
ALFWORLD-64	7.00	**14.00**	4.00	5.00	0.00
ALFWORLD-65	**25.00**	5.00	5.00	5.00	0.00
ALFWORLD-66	**18.00**	7.00	5.00	3.00	0.00
ALFWORLD-67	**5.00**	2.00	**5.00**	**5.00**	0.00
ALFWORLD-68	**25.00**	8.00	5.00	4.00	0.00
ALFWORLD-69	1.00	**20.00**	5.00	4.00	0.00
ALFWORLD-70	**23.00**	3.00	5.00	3.00	0.00
ALFWORLD-71	7.00	**10.00**	4.00	4.00	0.00
ALFWORLD-72	8.00	**18.00**	6.00	5.00	0.00
ALFWORLD-73	4.00	**8.00**	5.00	5.00	0.00
ALFWORLD-74	**26.00**	6.00	4.00	5.00	0.00
ALFWORLD-75	6.00	**25.00**	5.00	5.00	0.00
ALFWORLD-76	**21.00**	7.00	5.00	–	0.00
ALFWORLD-77	**7.00**	2.00	5.00	5.00	0.00
ALFWORLD-78	**28.00**	3.00	6.00	5.00	0.00
ALFWORLD-79	12.00	**16.00**	5.00	4.00	0.00
ALFWORLD-80	10.00	**15.00**	5.00	5.00	0.00
ALFWORLD-81	**11.00**	1.00	5.00	5.00	0.00
ALFWORLD-82	**5.00**	3.00	**5.00**	**5.00**	0.00
ALFWORLD-83	**8.00**	1.00	5.00	4.00	0.00
ALFWORLD-84	6.00	**10.00**	5.00	5.00	0.00
ALFWORLD-85	**7.00**	3.00	5.00	–	0.00
ALFWORLD-86	**33.00**	9.00	5.00	5.00	0.00
ALFWORLD-87	1.00	**7.00**	5.00	4.00	0.00
ALFWORLD-88	**28.00**	14.00	4.00	4.00	0.00
ALFWORLD-89	**12.00**	5.00	5.00	4.00	0.00
ALFWORLD-90	**27.00**	20.00	5.00	4.00	0.00
ALFWORLD-91	0.00	**8.00**	5.00	4.00	0.00
ALFWORLD-92	7.00	**10.00**	5.00	4.00	0.00
ALFWORLD-93	4.00	2.00	**5.00**	**5.00**	0.00
ALFWORLD-94	5.00	**11.00**	5.00	5.00	0.00
ALFWORLD-95	**14.00**	8.00	5.00	4.00	0.00
ALFWORLD-96	**23.00**	9.00	6.00	–	0.00
ALFWORLD-97	8.00	**10.00**	5.00	5.00	0.00
ALFWORLD-98	1.00	**13.00**	4.00	4.00	0.00
ALFWORLD-99	**24.00**	4.00	5.00	4.00	0.00
ALFWORLD-100	5.00	**13.00**	6.00	4.00	0.00

Table 9: Cumulative return per ALFWorld task (101–134). Higher ↑ is better. Single-seed runs (no CI).

Environment	LWM-Planner ↑	ReAct + FEC ↑	ReAct ↑	Reflexion ↑	Random ↑
ALFWORLD-101	7.00	**20.00**	5.00	4.00	0.00
ALFWORLD-102	**30.00**	4.00	5.00	5.00	0.00
ALFWORLD-103	4.00	**12.00**	4.00	4.00	0.00
ALFWORLD-104	**9.00**	7.00	5.00	4.00	0.00
ALFWORLD-105	**11.00**	10.00	5.00	4.00	0.00
ALFWORLD-106	**8.00**	5.00	5.00	–	0.00
ALFWORLD-107	**17.00**	3.00	5.00	4.00	0.00
ALFWORLD-108	–	**14.00**	4.00	4.00	0.00
ALFWORLD-109	**10.00**	2.00	5.00	4.00	0.00
ALFWORLD-110	9.00	**12.00**	5.00	5.00	0.00
ALFWORLD-111	**24.00**	7.00	6.00	–	0.00
ALFWORLD-112	**14.00**	10.00	5.00	4.00	0.00
ALFWORLD-113	**7.00**	1.00	6.00	–	0.00
ALFWORLD-114	6.00	**9.00**	5.00	4.00	0.00
ALFWORLD-115	0.00	**10.00**	5.00	–	0.00
ALFWORLD-116	**22.00**	6.00	5.00	4.00	0.00
ALFWORLD-117	0.00	**28.00**	5.00	4.00	0.00
ALFWORLD-118	0.00	**8.00**	5.00	5.00	0.00
ALFWORLD-119	2.00	**12.00**	5.00	–	0.00
ALFWORLD-120	5.00	**7.00**	6.00	5.00	0.00
ALFWORLD-121	**28.00**	12.00	5.00	4.00	0.00
ALFWORLD-122	3.00	**7.00**	5.00	5.00	0.00
ALFWORLD-123	**10.00**	4.00	5.00	4.00	0.00
ALFWORLD-124	4.00	4.00	**5.00**	4.00	0.00
ALFWORLD-125	4.00	**15.00**	6.00	4.00	0.00
ALFWORLD-126	**22.00**	9.00	4.00	4.00	0.00
ALFWORLD-127	**20.00**	2.00	4.00	5.00	0.00
ALFWORLD-128	6.00	**13.00**	4.00	5.00	0.00
ALFWORLD-129	0.00	**12.00**	4.00	5.00	0.00
ALFWORLD-130	0.00	**13.00**	5.00	4.00	0.00
ALFWORLD-131	**18.00**	4.00	5.00	3.00	0.00
ALFWORLD-132	**8.00**	6.00	5.00	5.00	0.00
ALFWORLD-133	3.00	**9.00**	5.00	4.00	0.00
ALFWORLD-134	18.00	**30.00**	5.00	4.00	0.00

J.2 Ablation Study - LWM-Planner Variants

We investigate the impact of the depth d, and branching factor b in our method on our main table of environments presented. We find that the ablations reveal that we fit to the text frozen lake searching MDP environment that having a depth of $d = 3$ and $b = 4$ performs best, which validates our initial choice of parameters.

Table 10: Cumulative return (0 = Random, 100 = Expert/best). Higher ↑ is better.

Method	alfworld_task_3 ↑	alfworld_task_5 ↑	alfworld_task_90 ↑	crafter_mini_5_s_0 ↑	grid_4_h_9_s_0 ↑
LWM-Planner (d=3, b=4)	15.33±39.62	21.00±228.71	**34.67±85.20**	119.33±124.34	**32.00±nan**
LWM-Planner (d=3, b=2)	1.33±3.79	**29.33±44.81**	11.00±14.90	**334.00±17.91**	15.00±25.41
LWM-Planner (d=1, b=4)	**27.00±60.29**	5.50±6.35	15.00±26.29	294.00±327.46	6.00±165.18
LWM-Planner (d=2, b=4)	1.50±6.35	19.50±120.71	12.67±12.25	217.67±249.46	0.00±nan

J.3 Main Table Results Un-Normalized

Table 11: Cumulative return (**higher better**) and steps per success (**lower better**); mean ± 95% CI, for each benchmark method across each environment. Bold indicates the best performing method for that metric and environment. Results are averaged over ten random seeds.

Method (metric)	TextFrozenLake (4×4; h=0.9)	CrafterMini (5×5)	ALFWorld-A	ALFWorld-B	ALFWorld-C
LWM-Planner (Cum. return ↑)	**31.80±20.39**	**150.30±44.94**	**21.33±9.53**	**22.89±12.11**	**19.50±8.37**
(Steps/Success ↓)	**6.00±0.00**	46.50±7.32	**8.44±1.46**	7.56±0.97	**7.55±1.10**
ReAct + FEC (Cum. return ↑)	20.20±12.19	149.70±55.50	4.70±2.46	15.50±6.51	10.60±3.71
(Steps/Success ↓)	–	**41.35±5.72**	14.55±12.27	**5.75±2.87**	9.35±5.35
ReAct (Cum. return ↑)	-265.20±33.59	92.00±57.16	12.60±0.37	12.80±0.45	12.50±0.38
(Steps/Success ↓)	–	50.70±5.47	24.70±0.96	23.80±1.29	25.05±0.52
Reflexion (Cum. return ↑)	-61.10±4.80	87.20±51.45	11.00±0.00	11.00±0.45	11.33±0.38
(Steps/Success ↓)	23.20±3.97	80.05±39.46	25.67±0.77	26.19±0.77	25.94±0.95
Random (Cum. return ↑)	-80.00±4.49	-289.00±8.56	0.00±0.00	0.00±0.00	0.00±0.00
(Steps/Success ↓)	–	–	–	–	–

J.4 Main Table Success Rates

Table 12: Environment-normalised success-rate on the five main benchmarks. For each environment the cumulative return of every method is divided by the *largest* return obtained on that environment (always LWM-Planner here) and expressed as a percentage. Cells show mean ± 95 % CI over ten seeds; higher ↑ is better. Bold indicates the best performance per environment.

Method	TextFrozenLake (4×4)	CrafterMini (5×5)	ALFWorld-A	ALFWorld-B	ALFWorld-C
LWM-Planner	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**
ReAct + FEC	63.5±38.3	99.6±36.9	22.0±11.5	67.7±28.4	54.4±19.0
ReAct	0.0±0.0	61.2±38.0	59.1±1.7	55.9±2.0	64.1±1.9
Reflexion	0.0±0.0	58.0±34.2	51.6±0.0	48.1±2.0	58.1±1.9
Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0

J.5 ALFWorld Final Success Rates

We report the final success rate (FSR) on the complete 134-task ALFWorld evaluation suite. Although simple planners such as ReAct achieve high task coverage when given sufficient attempts, they do so with much lower quality roll-outs; LWM-Planner nearly matches their FSR while more than doubling the aggregate return. This highlights the practical value of investing additional computation to obtain higher-quality plans, a trade-off we further quantify in Appendix J.6.

Table 13: Environment-normalised success-rate on the main benchmark suite (TextFrozenLake, CrafterMini, ALFWorld-A/B/C). Higher ↑ is better. Each cell shows the mean over the five environments ± 95 % CI.

Method (metric)	Aggregate Normalised SR
LWM-Planner (Norm. SR ↑)	**100.0±0.0**
ReAct + FEC (Norm. SR ↑)	61.4±24.4
ReAct (Norm. SR ↑)	48.1±23.7
Reflexion (Norm. SR ↑)	43.1±21.5
Random (Norm. SR ↑)	0.0±0.0

Table 14: Final success rate (FSR; **higher better**) on the full 134-task ALFWorld evaluation suite together with the aggregate cumulative return (ACR). Values are mean ± 95% CI over three seeds.

Method	**FSR (%) ↑**	**ACR ↑**
LWM-Planner	**91.8±4.0**	**10.42±1.43**
ReAct + FEC	98.5±2.5	8.53±0.93
ReAct	**100.0±1.4**	5.00±0.09
Reflexion	92.5±4.2	4.36±0.10

J.6 Computational Cost Analysis

Table 15: Computational cost per environment step on ALFWorld under a 300-step interaction budget. Token counts are totals over the run.

Method	**Avg. tokens/call**	**Total tokens**	**Latency (s)**
LWM-Planner	≈993	827,931	58.6
ReAct + FEC	1,240	74,419	12.9
ReAct	1,045	35,530	11.1
Reflexion	749	47,190	14.2

J.7 Generalization to a Smaller Backbone (GPT-4o-mini)

To assess whether the gains of LWM-Planner arise from the planning procedure itself rather than only from a stronger backbone, we repeat the comparison with GPT-4o-mini. Table 16 reports the same normalised cumulative-return metric used in the main comparison. Although performance degrades across methods under the smaller backbone, the overall pattern remains consistent: simple reactive prompting remains weak, fact memory alone is already competitive on some shorter-horizon settings, and grounded lookahead remains most useful on the harder trap-filled environments where planning errors compound.

Table 16: Normalised cumulative return (**higher better**) with a GPT-4o-mini backbone across the five benchmark environments. Values are mean ± 95% CI over three seeds.

Method	TextFrozenLake (4×4; h=0.9)	CrafterMini (5×5)	ALFWorld-A	ALFWorld-B	ALFWorld-C
ReAct	-84.00±5.69	-83.60±45.53	3.40±1.11	**0.00±0.00**	4.20±1.04
ReAct + FEC	-3.40±7.38	4.80±72.82	**11.40±7.68**	**0.00±0.00**	4.60±8.22
LWM-Planner (d=1)	1.33±15.97	7.50±47.78	3.50±11.14	**0.00±0.00**	**9.00±6.02**
LWM-Planner (d=2)	**3.00±2.14**	4.60±8.81	2.20±2.96	**0.00±0.00**	7.40±6.95
Reflexion	-50.20±2.69	**10.00±64.54**	2.80±1.36	**0.00±0.00**	1.20±1.36
Random	-27.80±3.21	-100.00±0.00	0.00±0.00	**0.00±0.00**	0.00±0.00

Table 17: Average tokens per action and normalised cumulative return (**higher better**) on TextFrozenLake (4×4; h=0.9). Values are mean ± 95% CI over three seeds.

Method	Avg. tokens/action	Cum. return norm ↑
ReAct	449.92±6.32	-50.00±6.57
Reflexion	477.83±10.83	-21.33±7.17
ToT (d=1)	514.50±5.53	-52.33±24.51
ReAct + FEC	1,162.05±13.06	2.33±17.62
ToT (d=2)	1,531.36±11.80	-57.67±10.04
ToT (d=3)	3,573.17±27.52	-61.67±2.87
LWM-Planner (d=2)	3,999.00±132.03	**8.00±8.61**
ToT (d=4)	4,391.14±101.57	-60.67±2.87
LWM-Planner (d=1)	4,421.50±94.98	4.67±11.74
ToT (d=5)	4,944.88±126.34	-51.33±2.87
RAP (d=1)	6,834.75±212.25	-100.00±0.00
RAP (d=2)	13,669.50±424.50	-83.33±71.71
RAP (d=3)	20,504.25±636.74	-55.33±96.09

On the easier and shorter-horizon settings, ReAct + FEC remains competitive, consistent with the observation that fact memory alone can already capture much of the useful task structure. In contrast, on TextFrozenLake, where avoiding hidden traps requires grounded forward reasoning, LWM-Planner remains the only variant that preserves positive return under the smaller backbone. This indicates that the improvement is not solely a consequence of model scale: the method improves the reasoning process itself through fact-conditioned planning.

J.8 Compute-Performance Pareto Front

We next examine whether simply scaling inference-time compute is sufficient in this interactive setting. Table 17 reports the primary compute–performance frontier on TextFrozenLake, measured by average tokens per action and normalised cumulative return. Raw search baselines consume substantially more tokens as depth increases, but their returns remain negative, indicating that additional search alone does not resolve model errors in partially observable environments. In contrast, fact-grounded lookahead converts extra compute into better decisions, yielding the only positive-return regime in this comparison.

Table 18 provides a supplementary success-efficiency view of the same experiment, reporting steps per success where successful episodes were observed. The strongest search-only baselines still fail to translate higher token budgets into reliable task completion, whereas both LWM-Planner variants and the ReAct + FEC ablation attain the optimal six-step solution whenever they succeed.

Taken together, these tables show that the gains are not explained by brute-force token usage alone. Simply allocating more inference-time compute to raw search saturates at poor performance, whereas the fact-conditioned planner attains better returns at substantially lower cost than the highest-compute baselines. The practical effect is a shift in the cost-performance frontier rather than a uniform increase in spending.

J.9 Adversarial Filter Audit

We further stress-test the predictive-consistency filter against "false but simplifying" hallucinations. Starting from a TextFrozenLake trajectory that terminates in a hole, we inject candidate facts and evaluate whether each

Table 18: Normalised cumulative return (**higher better**) and steps per success (**lower better**) on TextFrozenLake (4×4; h=0.9). Values are mean ± 95% CI over three seeds.

Method (metric)	TextFrozenLake (4×4; h=0.9)
ToT (d=1) (Cum. return norm ↑)	-52.33±24.51
(Steps/Success ↓)	–
ToT (d=2) (Cum. return norm ↑)	-57.67±10.04
(Steps/Success ↓)	–
ReAct + FEC (Cum. return norm ↑)	2.33±17.62
(Steps/Success ↓)	**6.00±0.00**
LWM-Planner (d=2) (Cum. return norm ↑)	**8.00±8.61**
(Steps/Success ↓)	**6.00±0.00**
RAP (d=1) (Cum. return norm ↑)	-100.00±0.00
(Steps/Success ↓)	–
RAP (d=3) (Cum. return norm ↑)	-55.33±96.09
(Steps/Success ↓)	–
LWM-Planner (d=1) (Cum. return norm ↑)	4.67±11.74
(Steps/Success ↓)	**6.00±0.00**
ReAct (Cum. return norm ↑)	-50.00±6.57
(Steps/Success ↓)	–
Reflexion (Cum. return norm ↑)	-21.33±7.17
(Steps/Success ↓)	51.67±57.74
Random (Cum. return norm ↑)	-32.00±4.30
(Steps/Success ↓)	–
RAP (d=2) (Cum. return norm ↑)	-83.33±71.71
(Steps/Success ↓)	–
ToT (d=3) (Cum. return norm ↑)	-61.67±2.87
(Steps/Success ↓)	–

candidate would be retained by a stronger trajectory-grounded audit against the recorded outcome. This isolates whether comfortable but incorrect world-model edits could pass the filter.

This audit shows that the filter does not reward simplification for its own sake. Because the audit is anchored to the recorded trajectory, even comfortable hallucinations such as asserting safety on a dangerous tile incur high prediction error and are rejected, while only the trajectory-consistent fact is retained.

Table 19: Adversarial filter audit on TextFrozenLake (4×4; h=0.9). We inject candidate facts into a trajectory ending in a hole and report the model prediction under the injected fact together with the filter decision. Only the trajectory-consistent fact is accepted.

Fact Type	Example Injection	Model Prediction w/ Fact	Filter Decision
None	(No facts)	High Error (Assumes Safe)	N/A
True Fact	"(0,2) is a hole"	Low Error (Predicts -1)	**Accept**
Hallucination	"(0,2) is a goal"	Very High Error (Predicts +1)	**Reject**
Safety Bias	"(0,2) is safe"	Very High Error (Predicts 0)	**Reject**
Irrelevant	"The sky is blue"	No Change	**Reject**

K Discussion of Limitations

While LWM-Planner demonstrates a promising approach to enhancing LLM agent planning capabilities through in-context learning via atomic fact augmentation and lookahead search, it is important to acknowledge several limitations. These limitations, detailed below, also point towards avenues for future research and refinement.

K.1 Fact Management and Quality

The efficacy of LWM-Planner is significantly predicated on the quality, relevance, and atomicity of the facts extracted by the f_θ component.

- **Quality and Relevance of Extracted Facts**: The process for extracting facts is guided by LLM-based interpretation of episodic trajectories (see Appendix B.1). While the aim is to identify "minimal new atomic facts" critical for improving future predictions (as per the motivation in Section 2.4), the current mechanism relies on the LLM's heuristic understanding. There is no formal guarantee that the extracted facts are indeed optimally atomic, critical, or non-redundant with prior knowledge or the environment description. Suboptimal facts could lead to inefficient use of the context window or, in worse cases, mislead the planning process, thereby affecting the practical realization of minimizing ϵ_{sim} and δ_{model}.
- **Scalability of Fact Memory**: The set of atomic facts, $\mathcal{M}_t$, is managed using a deque and an optional LLM-based compression step (Appendix B.1). However, in very long-running deployments or exceedingly complex environments, the number of unique, relevant facts might still grow substantially. This could eventually strain the LLM's context window capacity, potentially leading to a performance bottleneck or the loss of older, still relevant facts if the deque's maximum length is exceeded or compression is overly aggressive. The impact on achieving a small $|\mathcal{Z}_{\mathcal{F}}| \ll |\mathcal{S}|$ (Section 2.1) in such scenarios needs further investigation.
- **Nature of Atomic Facts**: The current framework operates on textual atomic facts. While flexible, this lacks a formal grounding typically found in symbolic AI systems where predicates have precise semantics and grounding in an ontology or logical theory. The definition of "atomic" is operational (minimal useful textual statements) rather than tied to a formal decomposition of the state space or transition dynamics. This could limit the systematicity and verifiability of the learned knowledge.

K.2 Planning and Simulation

The lookahead search mechanism, while powerful, also introduces certain limitations.

- **Computational Cost**: The recursive lookahead search (Algorithm 2) involves multiple LLM calls for action proposal (g_ϕ^{propose}), state simulation (g_ϕ^{simulate}), and value estimation (g_ϕ^{value}) at each search node. The computational cost can therefore be considerable, scaling with search depth (D_s) and branching factor (k_B). While memoization within a single planning step helps (as mentioned in Section 3), the overall latency might be prohibitive for environments requiring very rapid decision-making. Future work could explore how to hybridize some of these components to reduce the computational burden.

- **Fidelity of LLM-based World Model and Value Function**: The core assumption is that an LLM, augmented with relevant atomic facts, can serve as an accurate latent world model (minimizing δ_{model}) and a reliable value estimator (contributing to minimizing ϵ_{plan}). While LLMs have shown impressive reasoning capabilities, their simulations of environmental dynamics or estimations of long-term value can be imperfect, especially in novel situations not well-covered by the current fact set or for states requiring deep causal reasoning beyond the LLM's inherent capabilities. Errors in simulation or value estimation can directly lead to suboptimal planning.
- **Fixed Search Parameters**: The current LWM-Planner employs a fixed search depth (D_s) and branching factor (k_B). This is a simplification, as optimal search effort can vary significantly depending on the current state's complexity or uncertainty. A more adaptive search control mechanism, potentially guided by confidence scores from the LLM components, could improve both performance and efficiency.

K.3 In-Context Learning Constraints

The reliance on in-context learning, while avoiding weight updates, has its own set of challenges (Dong et al., 2024).

- **Context Window Capacity**: The primary constraint is the finite context window of current LLMs. All learned knowledge (atomic facts) and recent interaction history must fit within this window to inform the LLM's operations. This inherently limits the total amount of experience that can be directly brought to bear at any single decision point or during fact extraction.
- **Knowledge Retention and "Forgetting"**: The management of the atomic fact set via a deque and optional compression aims to keep the most relevant information. However, there's a potential for "forgetting" older facts that might still be crucial if they are pushed out of the deque or overly compressed. The efficacy of the LLM-based compression in preserving all and only essential information is heuristic.
- **Rate of Learning and Adaptation**: Learning occurs implicitly through the curation and augmentation of the fact set. While this allows for online adaptation, the rate of learning or the ultimate performance ceiling might be constrained by the LLM's inherent in-context learning capabilities compared to methods that can fine-tune model weights on accumulating experience.

K.4 Theoretical Framework and Assumptions

The theoretical motivation in Section 2 provides a valuable formal basis but relies on certain idealizations.

- **Idealized Abstraction**: The framework assumes the possibility of an ϵ_{sim}-approximate bisimulation via fact-based abstraction Ψ^*. In practice, the LLM-driven fact extractor f_θ approximates this ideal, and the quality of this approximation directly impacts the bounds. Achieving and verifying such a bisimulation with textual facts is an open challenge.
- **Perfect Abstract Model Assumption (Initially)**: Theorem 2.4 assumes a perfect model M_{Ψ^*} of the abstract MDP. While Equation (6) accounts for model learning error δ_{model}, the practical estimation and minimization of this error when the model is implicitly defined by an LLM conditioned on facts are complex.
- **Measurability**: Directly measuring quantities like ϵ_{sim} or δ_{model} for the LWM-Planner in practical settings is difficult, making it challenging to empirically verify the tightness of the derived theoretical bounds.

K.5 Broader Considerations and Future Work

- **Dependence on Foundational LLM Capabilities**: The performance of LWM-Planner is intrinsically linked to the capabilities of the chosen LLM (e.g., its reasoning, simulation, and instruction-following fidelity). Limitations in the base LLM will propagate to the agent (Hager et al., 2024).
- **Prompt Sensitivity**: Like many LLM-based systems, the performance of LWM-Planner's components can be sensitive to the precise phrasing and structure of the prompts (examples in Appendix B and C). Ensuring robustness and generalizability of these prompts across diverse tasks and environments may require significant engineering or meta-learning.

- **Generalization to Diverse Environments**: The current empirical evaluation focuses on text-based environments. Extending LWM-Planner to handle environments with continuous state/action spaces, partial observability, or multi-modal inputs would require adaptations, particularly in how atomic facts are defined, extracted, and utilized by the LLM components.
- **Exploration-Exploitation Balance**: LWM-Planner's current lookahead search is primarily geared towards exploitation of its current knowledge (facts and LLM capabilities). A more explicit mechanism for exploration, perhaps by using uncertainty in LLM-generated values or simulations to guide the search towards informative regions or to trigger targeted fact-finding actions, could further enhance learning and performance.
- **Cost of LLM Usage**: The reliance on multiple LLM calls, especially within the lookahead search, can lead to significant computational and API costs, which might be a practical concern for widespread deployment or very long-running experiments.

Addressing these limitations offers rich avenues for future research, potentially leading to even more robust, efficient, and broadly applicable LLM agents capable of sophisticated online learning and planning.

L Ethical Considerations and Broader Impact

The LWM-Planner framework, while aimed at advancing AI planning capabilities, introduces several ethical considerations and potential broader impacts that warrant careful discussion. Our approach relies on Large Language Models (LLMs) for core functionalities such as fact extraction, latent world model simulation, and value estimation. Consequently, it inherits both the strengths and weaknesses inherent in current LLM technology (Gallegos et al., 2024; Hager et al., 2024).

L.1 Ethical Considerations

- **Factual Accuracy and Reliability of Learned Knowledge**: A core component of LWM-Planner is the extraction and utilization of "atomic facts." The veracity and relevance of these facts are paramount. If the Ψ_{LLM} component (Fact Extractor) erroneously extracts incorrect facts or misinterprets trajectory data, the agent's world model and subsequent planning can become flawed. This could lead to suboptimal or even detrimental behavior, particularly if the agent is deployed in safety-critical applications. While our approach aims for "atomic" and verifiable facts, the LLM's generation process is not infallible, and mechanisms for fact validation and retraction may be necessary for robust real-world deployment.

- **Bias in LLM Components**: The underlying LLMs (g_ϕ and Ψ_{LLM}) are pre-trained on vast datasets, which may contain societal biases. These biases could manifest in how facts are interpreted or generated, how states are valued, or how actions are proposed and simulated. For example, an LLM might exhibit biases in simulated interactions involving representations of different demographic groups if such biases were present in its training data. This could lead to unfair or inequitable agent behavior if deployed in human-interactive settings. Ongoing research into bias detection and mitigation in LLMs is crucial for addressing these concerns.

- **Autonomy, Control, and Oversight**: LWM-Planner enhances agent autonomy by enabling online, in-context learning and adaptation without direct weight updates. While this is a research goal, increased autonomy necessitates robust mechanisms for human oversight, control, and the ability to intervene if the agent learns undesirable facts or behaviors. The "atomic facts" provide a degree of transparency into the agent's learned knowledge, which can aid in debugging and oversight, but ensuring safe and aligned behavior in complex, long-horizon tasks remains a significant challenge.

- **Computational Resources and Environmental Impact**: Training and deploying large-scale LLMs, even without fine-tuning for each task, requires significant computational resources, contributing to energy consumption and environmental concerns. While LWM-Planner aims for sample efficiency in terms of environment interactions, the LLM inference calls during planning (especially with lookahead search) can be computationally intensive. Future work should consider the efficiency of the planning and fact management processes to mitigate these impacts.

- **Explainability and Trust**: While the use of explicit "atomic facts" is intended to make the agent's reasoning more transparent than end-to-end black-box models, the internal decision-making of the LLM components themselves (e.g., how g_ϕ^{simulate} predicts a next state based on facts and observation) remains complex. Building trust in such systems requires further advancements in methods for interpreting and explaining LLM-driven reasoning processes, even when augmented with symbolic facts.

L.2 Broader Impact

- **Advancing AI Planning and Adaptability**: This research contributes to developing more capable AI agents that can learn from experience in-context and adapt their plans in dynamic environments. This could have positive implications for various fields requiring sophisticated planning, such as logistics, robotics, personalized education, and scientific discovery, by enabling agents to more efficiently tackle complex, long-horizon tasks.

- **Reduced Dependence on Extensive Fine-Tuning**: The LWM-Planner's ability to learn online through fact augmentation reduces the need for repeated, task-specific fine-tuning of the base LLM. This can lower the barrier to applying powerful LLMs to new sequential decision-making problems, saving data and computational resources typically associated with training specialized models.

- **Potential for Misuse or Unintended Consequences**: As with any advanced AI technology, more autonomous and adaptive agents could potentially be misused if deployed without appropriate safeguards. An agent learning incorrect or malicious "facts" in an unconstrained environment could lead to undesirable outcomes. Furthermore, the increasing capability of autonomous agents raises long-term questions about their role in society and the workforce.
- **Scalability and Generalization Challenges**: While LWM-Planner shows promise, scaling the approach to vastly more complex, open-ended, or partially observable environments presents significant challenges. The manageability of the "atomic fact" base, the combinatorial explosion of lookahead search (even if depth-limited), and the LLM's ability to accurately simulate highly novel scenarios are areas requiring further research. Overcoming these challenges is crucial for realizing the broader positive impacts of such agents.
- **Interaction with Humans**: If LWM-Planner or similar agents are deployed in scenarios involving human interaction, the nature of fact extraction and utilization becomes particularly sensitive. Facts learned from human interactions must be handled with care to ensure privacy, fairness, and to avoid perpetuating harmful stereotypes or misinformation. The design of human-agent interaction protocols that allow for collaborative fact validation and refinement will be important.

Continued research, alongside open discussion and the development of robust safety and ethical guidelines, will be essential to navigate the challenges and harness the benefits of increasingly autonomous and adaptive LLM-based agents like LWM-Planner.

M Reporting, Budgets, and Reproducibility

M.1 Statistical Reporting Protocol

Seeds and confidence intervals. Unless explicitly stated otherwise, aggregate values are reported as

$$\text{mean} \ \pm \ 1.96\,\hat{\sigma}/\sqrt{n},$$

with n independent seeds and $\hat{\sigma}$ the sample standard deviation across seeds. Single-seed figures (if any) are shown *without* CIs and are labeled as such.

Normalization of cumulative return. When we report a *normalized* cumulative return for an environment E, we map a raw return R to

$$\text{Norm}(R; E) \ = \ 100 \cdot \frac{R - R_{\min}(E)}{R_{\max}(E) - R_{\min}(E)},$$

where $R_{\min}(E)$ is the mean return of the Random baseline on E (over the same seeds) and $R_{\max}(E)$ is the largest mean return achieved by any method on E in that block. By definition, $\text{Norm}(\cdot) \in [0, 100]$.

Environment vs. LLM budgets. All methods share the same environment interaction budget (300 steps unless stated). We also disclose LLM usage (calls, input/output tokens, wall-clock) per method to clarify test-time compute; see App. M.2.

Reproducibility note. Captions specify seeds n; decoding settings are shared across methods (App. P).

M.2 Compute Budget Disclosure & Fairness Assumptions

We distinguish **environment budget** (steps in the simulator) from **LLM budget** (inference compute):

$$\text{Calls}, \quad \text{TokIn}, \quad \text{TokOut}, \quad \text{WallClock}.$$

All main results match the *environment* budget across methods. Because tree search naturally issues more LLM calls than single-shot policies, we disclose LLM usage (calls/tokens/latency) alongside returns to contextualize quality–compute trade-offs.

Fairness stance. Equal-steps is the primary budget we optimize for (comparable interaction data). LLM compute disclosures are provided for transparency; compute-matched variants (token or time caps) are straightforward (see

formulas in App. N.3) and can be included if requested. Our qualitative conclusions do not rely on unreported compute.

M.3 Minimal Reproduction Config

```
agent:
  depth: 3
  branch: 4
  gamma: 0.99
  step_penalty: 0.01
  history_len: 51
  fact_budget_tokens: 1500
  thresholds:
    alpha_r: 1.0
    alpha_d: 1.0
    alpha_o: 1.0
    eta: 0.0
llm:
  temperature: 0.0
  max_output_tokens: 8512
prompt:
  truncate:
    keep_env_header: true
    min_history_items: 6
eval:
  steps_total: 300
  seeds: [0,1,2]
  envs:
    - text_frozen_lake_4x4_h0.9
    - crafter_mini_5
    - alfworld_eval_ood
```

N Methodological Details (Supplementary)

N.1 Predictive-Consistency Filter: Formal Definition & Pseudocode

Positioning. Below we formalize a stronger trajectory-grounded validation variant of the predictive-consistency filter, used for supplementary validation and the adversarial audit in Appendix J.9. The deployed agent in the main text uses the lighter model-internal heuristic described in Section 3.

Goal. Retain a candidate fact f only if conditioning the simulator/value on f *reduces* next-step predictive error on the just-finished episode $\tau = \{(o_t, a_t, r_t, o_{t+1}, d_{t+1})\}_{t=0}^{H-1}$.

Error metric. For each step t, we form LLM predictions with temperature 0:

$$\hat{r}_t,\ \hat{o}_{t+1},\ \hat{d}_{t+1} \quad \text{(with and without } f\text{)},$$

and define a per-step loss

$$\ell_t \ = \ \alpha_r\,|\hat{r}_t - r_t| \ + \ \alpha_d\,\mathbf{1}\{\hat{d}_{t+1} \neq d_{t+1}\} \ + \ \alpha_o\,\mathrm{dist}(\hat{o}_{t+1},\, o_{t+1}),$$

where dist is a token-level normalized edit distance between canonicalized observation strings (lowercased; digits/punctuation kept). We use α_r=1, α_d=1, α_o=1 by default. Aggregate episode loss is $\mathcal{L} = \frac{1}{H}\sum_t \ell_t$.

Decision rule. Let $\mathcal{L}^{(\neg f)}$ be the loss without conditioning on f and $\mathcal{L}^{(f)}$ with f. Accept f iff

$$\Delta\mathcal{L}(f) \equiv \mathcal{L}^{(\neg f)} - \mathcal{L}^{(f)} \geq \eta, \quad \eta \in [0,1] \text{ (default } \eta{=}0).$$

Ties ($\Delta\mathcal{L} = 0$): accept only if f is *novel* (not subsumed by an existing fact).

Efficiency. We cache per-step LLM calls from the initial pass; testing each f reuses most responses. In practice we batch candidates in groups of 8–16 and memoize (o_t, a_t, F) keyed by a 64-bit hash.

Algorithm 4: Predictive-Consistency Filter (temperature 0)

Input: episode τ, current facts F, candidates $\mathcal{C}$, thresholds $(\alpha_r, \alpha_d, \alpha_o, \eta)$
Output: accepted facts $\mathcal{A}$

Compute $\mathcal{L}^{(\neg f)}$ once by running g_ϕ^{simulate} on τ conditioned on F
$\mathcal{A} \leftarrow \emptyset$
for $f \in \mathcal{C}$ **do**
  if *is_subsumed(f,F)* **then**
    continue
  Compute $\mathcal{L}^{(f)}$ by rerunning only the changed nodes conditioned on $F \cup \{f\}$ (memoized)
  if $\mathcal{L}^{(\neg f)} - \mathcal{L}^{(f)} > \eta$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{f\}$
  else if $\mathcal{L}^{(\neg f)} - \mathcal{L}^{(f)} = 0$ ***and*** *is_novel(f,F)* **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{f\}$
return $\mathcal{A}$

N.2 Prompt Budgeting & Truncation Policy

We cap each planning call's prompt to $T_{\max}$ tokens. Ordering (kept, then truncated):

1. Environment description (fixed header; shortened to canonical bullet list of actions).
2. Current observation o_t.
3. Atomic facts F_t (top-K by $\Delta\mathcal{L}$ then recency; K chosen so that facts $\leq B_f$ tokens).
4. Recent history: last H_L items; if over budget, elide with "…" and keep last $h_{\min}$ observations and actions.

On overflow we first shrink history, then facts (keeping higher-utility ones), never the environment header.

N.3 Search Cost Model (Calls and Tokens)

For depth D_s and branching k_B:

$$\#\text{nodes} = \sum_{i=0}^{D_s} k_B^i, \quad \#\text{edges} = \sum_{i=0}^{D_s-1} k_B^{i+1}.$$

Per root decision:

$$\begin{aligned} \#\text{propose} &= \sum_{i=0}^{D_s-1} k_B^i, \\ \#\text{simulate} &= \sum_{i=1}^{D_s} k_B^i, \\ \#\text{value} &= k_B^{D_s} \quad \text{(leaf nodes)}. \end{aligned}$$

Let average input tokens per call be (τ_p, τ_s, τ_v); expected tokens per planned action are

$$\mathbb{E}[\text{tokens}] \approx \tau_p \sum_{i=0}^{D_s-1} k_B^i \;+\; \tau_s \sum_{i=1}^{D_s} k_B^i \;+\; \tau_v \, k_B^{D_s}.$$

Memoization reduces effective factors by $\rho \in (0, 1]$ (empirically $\rho \approx 0.6$ for `simulate`); substitute $\tau_s \leftarrow \rho\,\tau_s$ for estimates.

O Theory—Practical Proxies

One-step model error proxy. On a replayed trajectory $\tau = \{(o_t, a_t, r_t, o_{t+1}, d_{t+1})\}$, define

$$\mathrm{MAE}_r = \tfrac{1}{|\tau|} \sum_t |\hat{r}_t - r_t|, \quad \mathrm{Acc}_d = \tfrac{1}{|\tau|} \sum_t \mathbf{1}\{\hat{d}_{t+1} = d_{t+1}\},$$

$$\mathrm{EditObs} = \tfrac{1}{|\tau|} \sum_t \mathrm{NED}\big(\mathsf{canon}(\hat{o}_{t+1}), \mathsf{canon}(o_{t+1})\big),$$

where predictions $(\hat{r}_t, \hat{o}_{t+1}, \hat{d}_{t+1})$ come from the LLM simulator at temperature 0 and $\mathsf{canon}(\cdot)$ is a fixed textual canonicalizer. Lower is better for MAE_r and EditObs, higher is better for Acc_d.

Aliasing proxy. For the abstraction $z_t = (o_t, F_t)$, estimate

$$\mathrm{Aliasing}(z) = \mathbb{E}_z\big[\mathrm{Var}_{s:\Psi(s)=z}\big(V^\pi(s)\big)\big]$$

by substituting V^π with Monte-Carlo returns under the current policy. As a lighter surrogate, report mutual information $I(z_t; r_{t:t+k})$ for small k. These proxies are reporting tools that connect to the motivational bound; no claims of tightness are made.

P Baseline Configuration & Prompt Parity Guarantees

Model/decoding. All methods use the same base LLM; planning/simulation/value calls run at temperature 0.0; ReAct "thought" generation (where applicable) uses temperature 0.3. Maximum output tokens are identical across methods; the same stop-sequences are used.

Environment headers. Prompts for all agents begin with the same environment description. Differences arise only in method-specific sections (e.g., facts, lessons, or search metadata).

History formatting. Observation/action history is identically formatted (chronological, fixed templates). For search, simulated histories reuse the same template to reduce style-induced variance.

Randomness. Each run fixes: environment seed, agent RNG seed, and any sampling temperatures. No external retrieval sources are used.

Q Environments and Solvability

Q.1 TextFrozenLake Solvability Construction (Formalization)

Given grid size N and hole probability h for non-start/goal cells, we construct a board that is always solvable:

1. Sample a monotone Manhattan path P by shuffling $N-1$ "right" and $N-1$ "down" moves from $(0,0)$ to $(N-1, N-1)$.
2. Mark all cells on P as ice; set $S = (0,0)$, $G = (N-1, N-1)$.
3. For every other cell, independently sample Hole $\sim$ Bernoulli(h).

Guarantee. The path P remains hole-free by construction, so the instance is solvable. Observations reveal only local cell types; rewards are $+1$ at G, -1 on holes, 0 otherwise; episodes end on G, a hole, or at $8(N-1)$ steps.

R Validity, Diagnostics, and Oracle Specification

R.1 Threats to Validity & Decontamination Steps

(T1) Pretraining contamination. The LLM may contain knowledge of benchmarks. Mitigation: (i) eval-OOD split for ALFWorld; (ii) fact acceptance requires predictive gain on the *current* trajectory (App. N.1); (iii) environment descriptions avoid revealing full solution strings.

(T2) Determinism vs. robustness. Planning uses temperature 0; to guard against brittle choices we (a) penalize steps (λ_{step}), (b) prefer higher-utility facts, and (c) memoize expansions to avoid drift within a decision.

(T3) Compute parity. We match the step budget across methods; search methods naturally use more tokens. We therefore report both cumulative return and token/latency (App. N.3, J.6).

(T4) Metric sensitivity. We report cumulative return *and* (in appendix) success-rate variants; we clarify reward shaping per environment (see benchmark details).

R.2 Typical Failure Modes & Diagnostics

F1. Stale fact usage. The agent continues to trust a fact contradicted later. *Mitigation:* demote contradictions when detected; re-validate on next episode.

F2. Over-compression of facts. Compression prunes a low-frequency but critical fact. *Mitigation:* utility-aware pruning; maintain a floor $K_{\min}$ on kept facts.

F3. Myopic value at leaves. Value estimator underestimates deferred rewards. *Mitigation:* increase D_s one level or raise the value-call budget; optionally add a short roll-out at leaves.

F4. Proposal collapse. g_ϕ^{propose} returns near-duplicates. *Mitigation:* n-gram diversity penalties over action strings before expansion.

F5. Simulator drift. Simulated observations diverge stylistically, hurting edit distance. *Mitigation:* enforce a canonical observation template and post-process to that template before measuring textual distance.

R.3 Oracle Ablation Design (Specification; No New Results)

To bound the headroom from perfect one-step simulation, we define an *oracle* planner that replaces g_ϕ^{simulate} with the environment's true (o', r', d') for domains that expose a fast model step (TextFrozenLake, CrafterMini). We keep g_ϕ^{propose} and g_ϕ^{value} unchanged. This isolates the contribution of one-step fidelity (proxy δ_{model}). We expect the gap between LWM-Planner and Oracle to shrink as the fact set matures.