LUMINA: Long-horizon Understanding for Multi-turn Interactive Agents

Amin Rakhsha[*1,2] **Thomas Hehn**[3] **Pietro Mazzaglia**[3]
Fabio Valerio Massoli[3] **Arash Behboodi**[3] **Tribhuvanesh Orekondy**[3]

[1]University of Toronto [2]Vector Institute [3]Qualcomm AI Research[†]

Abstract

Large language models can perform well on many isolated tasks, yet they continue to struggle on multi-turn, long-horizon agentic problems that require skills such as planning, state tracking, and long context processing. In this work, we aim to better understand the relative importance of advancing these underlying capabilities for success on such tasks. We develop an oracle counterfactual framework for multi-turn problems that asks: how would an agent perform if it could leverage an oracle to perfectly execute a specific skill? The change in the agent's performance due to this oracle assistance allows us to measure the criticality of that skill in the future advancement of AI agents. We introduce a suite of procedurally generated, game-like tasks with tunable complexity. These controlled environments allow us to provide precise oracle interventions, such as perfect planning or flawless state tracking, and make it possible to isolate the contribution of each oracle without confounding effects present in real-world benchmarks. Our results show that while some interventions (e.g., planning) consistently improve performance across settings, the usefulness of other skills is dependent on the properties of the environment and language model. Our work sheds light on the challenges of multi-turn agentic environments to guide the future efforts in the development of AI agents and language models.

1 Introduction

Large Language Models (LLMs) have demonstrated exceptional performance across a wide range of tasks, including natural conversations (Touvron et al., 2023; Achiam et al., 2023), question answering (Yang et al., 2018), competitive coding (Austin et al., 2021; White et al., 2024), and mathematical reasoning (Comanici et al., 2025; Guo et al., 2025). Consequently, given their general-purpose capabilities, a natural question that is actively being explored is whether language models can be leveraged as *multi-turn* agents, i.e., whether they can iteratively perceive, reason, and take strategic actions towards achieving a distant goal. Such tasks introduce a host of new challenges, such as maintaining coherence over multiple turns and long contexts, reasoning over multiple dynamic pathways, recovering from errors, identifying the right tools for the task, and efficiently tracking state and progress without explicit feedback.

Numerous benchmarks quantitatively analyze multi-turn agent capabilities across various domains, ranging from function calling (Patil et al., 2025), web navigation (Koh et al., 2024), interactive coding (Trivedi et al., 2024), human interaction (Liu et al., 2023), and game-playing (Guertler et al., 2025). Ongoing results suggest that there is plenty of progress to be made, e.g., on AppWorld (Trivedi et al., 2024), GPT-4o has a success rate of 30.2% and open-weight models such as LLama3-70B (Grattafiori et al., 2024) achieve 7.0%. As many of the benchmark analyses report, pushing progress requires enabling numerous skills, such as efficient task decomposition, planning, state tracking, and information gathering. Even though the necessity of these skills has been individually established in prior work, their relative importance on the agent's success remains an open question.

In this paper, our goal is to understand which of these skills (or combination thereof) are the bottleneck to make progress towards capable multi-turn agents. To assess skill importance, we introduce an oracle intervention framework that poses counterfactual questions:*What if the agent had access to the information that each skill provides?* Using this framework, we investigate three oracle interventions and their combinations: *planning*, which provides a description of a single-turn subtask on

[*]Work completed during an internship at Qualcomm AI Research

[†]Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

Findings of the Association for Computational Linguistics: ACL 2026, pages 3913–3926

the optimal trajectory, *state tracking*, which provides an accurate state of the agent, and *history pruning*, which rewrites the context to remove distractors and provide a compact summary.

Measuring the impact of individual skills on an agent's success in multi-turn, goal-oriented tasks is challenging. Real-world ablations are cumbersome and often impractical because these tasks may admit numerous equally optimal paths to the goal, making exhaustive oracle annotations intractable. To overcome this, we introduce procedurally-generated game environments where optimal actions and strategies can be computed at any step, enabling systematic oracle interventions. We consider three game-playing multi-turn tasks: *ListWorld* to cautiously mutate a python list using only `pop` actions, *TreeWorld* to find a particular node in a tree iteratively using only `get_children` actions, and *GridWorld* to spatially navigate a 2d grid to reach a target location while avoiding holes. All environments are configurable, and importantly, enable us to inject accurate oracle information at any point of the agent's trajectory.

Our framework enables us to examine the capabilities of multi-turn agents along multiple dimensions (e.g., task complexity, model size, and the influence of specific oracle interventions) and helps us provide insights into **L**ong-horizon **U**nderstanding of **M**ulti-turn **IN**teractive **A**gents (**LUMINA**). First, we observe a significant discrepancy between the low success rates over long-horizon trajectories and the high per-step accuracy (i.e., whether the chosen action is from the set of optimal actions), indicating that compounding errors represent a major challenge (Sinha et al., 2025; Li et al., 2025). Second, we find that oracle interventions generally improve the success rate, but their effectiveness varies depending on model size. Finally, different environments have distinct characteristics that translate to varying gains from the interventions and measurable performance discrepancies between success rate and step accuracy. Overall, our findings present a double-edged picture: while improving specific skills (enabled by oracles interventions in our case) generally help the LLM-based agents solving multi-turn tasks, fully bridging the gap likely requires exploiting environment and model-specific understanding.

2 Related Works

LLMs and Agents. Agent-based systems, defined by a policy interacting with an environment to achieve goals and receive rewards (Russell et al., 1995), have a long history. Recent work shows that language models can serve as the policy (Huang et al., 2022; Yao et al., 2023b), the environment as a world model (Hao et al., 2023), or the reward function (Zheng et al., 2023; Zhang et al., 2025). We focus on LLMs as policies that auto-regressively sample next actions. A prominent example is ReAct prompting (Yao et al., 2023b), which interleaves reasoning and task-specific actions. ReAct has proven effective across domains, from game playing (Wang et al., 2023a) to interactive coding (Trivedi et al., 2024). In this work, we adopt ReAct-based prompting to elicit dynamic reasoning and planning from LLMs.

LLM Agent Capabilities. The are several essential skills for agents beyond language understanding and reasoning. Agents must execute admissible actions (e.g., tools, function calls) with correct arguments (Qin et al., 2023; Patil et al., 2024). As there may be multiple paths towards the goal, agents should support multi-path reasoning (Besta et al., 2024; Yao et al., 2023a) and dynamic re-planning (Song et al., 2023) to adapt to feedback. At each turn, effective decision-making can require both short-term (context window) and long-term memory (external storage) (Song et al., 2023; Huang et al., 2023; Wang et al., 2023b). Additionally, agents might need to track evolving hidden environment states influenced by prior actions (Ebrahimi et al., 2024; Vodrahalli et al., 2024). These, and potentially more, capabilities collectively underpin robust agentic behavior.

Characterizing Bottlenecks for Agents. Approaches to assessing model effectiveness in agentic tasks include holistic benchmarks (Trivedi et al., 2024; Patil et al., 2024) and capability-level analysis. Benchmarks show a clear trend: larger models outperform smaller ones, prompting the question—what drives these gaps? Few works exist that isolate bottlenecks in multi-turn tasks: reinforcement learning (RL) influence in strategic games like mazes (Abdulhai et al., 2023), error taxonomies for multi-agent systems in AppWorld (Cemri et al., 2025), and concurrent to our work, execution limits in long-horizon tasks (Sinha et al., 2025). In contrast, we study the bottlenecks in

procedurally-generated game environments, which allows us to inject oracle interventions.

3 Formulation: LUMINA

In this section, we begin by detailing the underlying process that requires an agent to perform sequential decision-making to complete the task. To better enable the agent to complete the task, we then elaborate on how to augment information at each turn using oracle interventions.

POMDP Tasks. We study tasks that can be modeled as a Partially-observable Markov Decision Process (POMDP): $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, \Omega, H, S_{\text{Goal}} \rangle$ where $\mathcal{S}$ is the hidden state space, $\mathcal{A}$ the agent's action space, and $\mathcal{O}$ is the observation space. Furthermore, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function (deterministic in our case) and $\Omega : \mathcal{S} \times \mathcal{A} \to \mathcal{O}$ is the observation function. The termination can be performed either by the agent (e.g., `DONE` action) or by the environment (e.g., $t \geq$ horizon H). For simplicity, we consider a terminal reward function: the agent receives a positive reward of $+1$ if it terminates at the goal state S_{Goal} within at most H steps and receives 0 elsewhere. The objective of the agent is to maximize the probability of success, which we refer to as the *success rate* and denote by $J(\pi_\theta)$ for an agent π_θ.

Base (ReAct) Policy Agent. Towards taking sequential decisions to solve the task (represented as text x), we consider a stochastic policy modeled by a ReAct (Yao et al., 2023b) LLM agent π_θ, where at each step t:

$$a_t \sim \pi_\theta \left(\cdot \mid x, h_{t-1} \right).$$

Here, $h_{t-1} := (a_1, o_1, ..., a_{t-1}, o_{t-1})$ is the history of the past interactions between the agent and the environment. Action a_t consists of both a chain-of-thought text (which is irrelevant to the environment) and one of the allowed operations.

Oracle Interventions. To better understand the relative role of each skill, we consider oracle interventions to assist policy π_θ by augmenting the observations with auxiliary information. We consider the case where policy π at every step is conditioned on the **oracle-augmented history** $\tilde{h}_t$:

$$a_t \sim \pi_\theta \left(\cdot \mid x, \tilde{h}_{t-1} \right),$$
$$\tilde{h}_{t-1} = \mathcal{O}^{\text{history}} \Big(h_{t-1} \oplus \mathcal{O}^{\text{plan}}(x, h_{t-1}) \oplus \mathcal{O}^{\text{state}}(x, h_{t-1}) \Big).$$

Generally, our oracle formulation accommodates appending ($\oplus$) a hint for the next step of an optimal plan (via $\mathcal{O}^{\text{plan}}$) and a summary of the belief state (via $\mathcal{O}^{\text{state}}$) to the history, as well as representing the context compactly by pruning the history of redundant information (via $\mathcal{O}^{\text{history}}$). We elaborate on the details of each of these in the following paragraphs.

Planning $\mathcal{O}^{\text{plan}}$. As apparent from the POMDP formulation, solving multi-turn environments requires reasoning about many steps into the future and devising a plan towards completing the task. At every step t, the planning intervention $\mathcal{O}^{\text{plan}}(x, h_{t-1})$ is the description of a single-turn subtask. This subtask is designed not to require planning (no reasoning about the future steps needed). Most importantly, the action that accomplishes this subtask is one of the optimal actions in the environment at that moment.

State Tracking $\mathcal{O}^{\text{state}}$. Solving partially-observable long-horizon tasks requires the agent to accurately track its knowledge about the hidden state of the environment at every step. This is highly challenging, since at each turn the agent needs to collect the information implicitly from its history of interactions with the environment and reason about the environment transitions. Consequently, we introduce an oracle $\mathcal{O}^{\text{state}}(x, h_{t-1})$ to present the precise state (e.g., current location in GridWorld) to the agent in a compact natural language form, where the state is determined from the agent's initial state and subsequent actions. As a result, the oracle here can be interpreted as enabling the agent to exploit Markov property in the POMDP task.

History Pruning $\mathcal{O}^{\text{history}}$. It is well-known that the performance of LLMs degrade as the size of the context (history h_t in our case) grows. More relevant to us, existing work (Laban et al., 2025; Vodrahalli et al., 2024) highlights that the performance at the same task drops merely due to the presence of distractors. This is a common problem in multi-turn LLM agentic tasks, where the history contains overcomplete information to guide the agent towards the goal. Since the size of the context (history h_t) grows at each turn, this makes decision-making more error prone. To mitigate the influence of distractors, we consider an oracle $\mathcal{O}^{\text{history}}$ that reduces the contents of the context into a compact form. In this work, we consider the simple implementation that can be done when

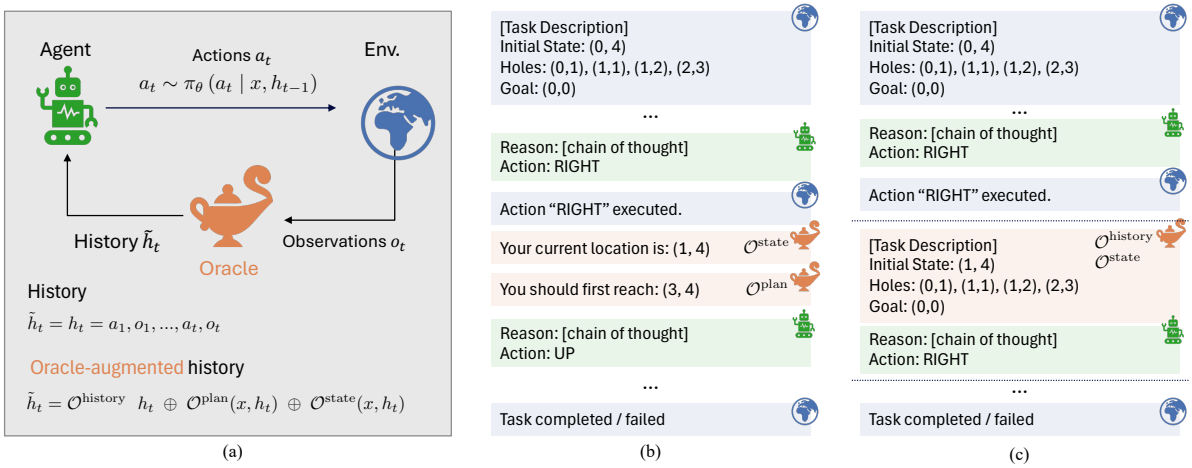


Figure 1: Formulation. (a) Oracle-augmented history. Within multi-turn tasks, we study LLM-based agents π_θ when additionally assisted by an oracle. We can leverage one or more oracles to modify the history h_t (context for language model). **(b) GridWorld example.** In this example, the agent needs to navigate from an initial 2D location to a goal location. We can use oracle $\mathcal{O}^{\text{state}}$ to summarize the current location (instead of the model reflectively reasoning at each turn). Similarly, we can also use $\mathcal{O}^{\text{plan}}$ to hint way points to reach the goal. **(c) History pruning**. Since we consider Markov decision processes, $\mathcal{O}^{\text{history}}$ can be used to rewrite the task description such that the actions can be taken independent to previous steps.

state tracking is present. In this case, we drop the old history h_{t-1}, since when the compact summary $\mathcal{O}^{\text{state}}(x, h_{t-1})$ is given, h_{t-1} is not necessary.

Step vs Task Metric J_{step}. In multi-turn environments, the success rate can often be low because agents are required to take the correct action at each step consistently. This challenge is compounded by the unique difficulties inherent in such environments, such as planning and state tracking. For example, even if the environment involves solving a sequence of single-turn reasoning problems, the performance can suffer due to the dependency on the chain, irrespective of the absence of planning. However, there are cases where this setup can be advantageous by offering the agent retries; for instance, in scenarios where solving a fraction of problems is enough for success. To measure this aspect, we define an objective over each step. We consider a step accurate if the action taken is optimal for that step. We refer to this metric as *step accuracy*.

4 Experimental Results

In this section, we first present the environments and tasks used in our experiments, the agent and oracle setups, and conclude by reporting our empirical results and insights.

4.1 Procedurally-generated Multi-turn Environments

Our goal is to characterize bottlenecks of multi-turn agents by leveraging oracle interventions. Existing benchmarks fall short for this task, since data are predominantly human-generated and are rarely accompanied with trajectory-level annotation. Some works (Trivedi et al., 2024) investigate marginal gains through oracle interventions, however in a very narrow scope that is admissible within the dataset. Consequently, we propose procedurally-generated multi-turn environments with the following requirements: (a) Minimal external knowledge: such that all necessary information can be specified in the prompt; (b) Simple action space: to prevent failures from constructing complex function calls; (c) Variable complexity: to enable us analyze success by varying the complexity of the task in a procedural manner; (d) Compositionality: such that tasks can be programmatically and accurately broken down into clear subproblems; (e) No data contamination: since the tasks are novel and can additionally be randomly re-generated, there is little risk from contamination; and most importantly, (f) Oracle interventions: since we know the underlying process at any instant, we can faithfully construct oracle interventions. Figure 2 shows partial examples of our environments.

General Framework. All our environments can be cast as a POMDP. Given an initial task x that can be communicated verbally, the agent needs to complete the task with a finite turn budget $T_{\max} \geq mT^* + n$, where T^* is the number of actions required by an optimal policy. To achieve the goal, the agent interacts with the environments using a simple set of actions. All environments

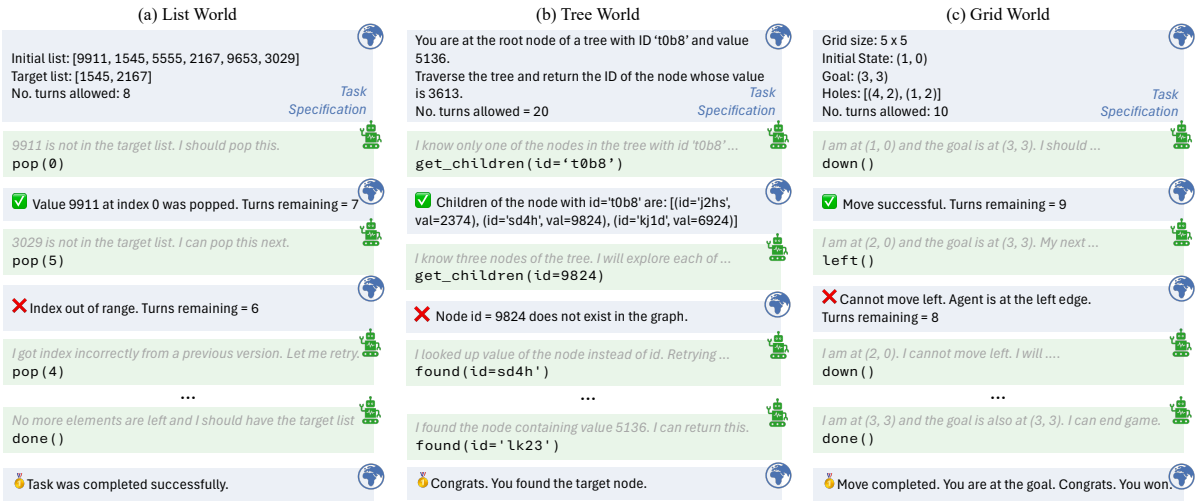


Figure 2: Environments. In this work, we study the influence of oracle interventions in three unique environments. In all cases, the agent reasons (shown in gray italic) and performs an action (shown in monospace), and the environment provides minimal but sufficient feedback to help the agent progress towards the goal. **(a) ListWorld**: which requires modifying a python list using only `pop(idx)` actions; **(b) TreeWorld**: where the task is to iteratively search over a tree to find a specific node; and **(c) GridWorld**: where the agent needs to move from an initial location to a goal location.

have a common terminating action `done`, which the agent needs to invoke once it completes the objective. The environment provides minimal essential feedback (e.g., 'move successful') at each turn.

ListWorld. Inspired by Vodrahalli et al. (2024), we introduce ListWorld to evaluate the ability of an LLM agent to sequentially modify and track the state of an initial object. Specifically, the task of the agent is to prune an initial `input` Python list to a smaller `target` list. The agent needs to prune using a single action: `pop(index)`. The agent has to pop the elements from left to right: once an element is popped, it becomes illegal to pop the elements before it. We control the task complexity by varying the number of elements that need to be pruned (i.e., `len(initial) - len(target)`). To complete the task successfully, the agent at each turn needs to: (a) determine current list by accounting initial list and historical actions; (b) find the candidates to prune; and (c) pop the corresponding candidate. This introduces a subtle challenge of understanding partial changes to the index-value mappings after every successful pop operation.

TreeWorld. In this environment, we study an agent's ability to sequentially explore and gather information at each turn. Specifically, the task is tree traversal: the agent needs to navigate from a source node to a target node that has a particular value. The nodes (except source and target) and edges are unknown to the agent. For simplicity and ease of analysis, we consider traversing from the root to a leaf node of a tree. The agent needs to traverse the tree using a single action: `get_children(node_id)`. Efficiently completing this task requires the agent to keep track of the frontier of unexplored nodes and sequentially explore them. We vary the complexity of the task by controlling the number of nodes in an m-ary tree.

GridWorld. We also consider a 2D grid world environment to study an agent's ability to plan and spatially navigate towards a goal. Our grid world takes the form of an $N \times N$ grid with holes, where stepping into the holes incurs an additional cost. The environment is fully observable, with each task stating the agent's start position and the goal position. The agent needs to navigate to the goal using one of four actions (`up()`, `down()`, `left()`, or `right()`) and reach the goal within a specified cost budget.

4.2 Setup: LLM Agents

In this section, we walk through the models we used for evaluation, how the prompts were designed to elicit closed-loop interactions with the environment, and also discuss evaluation metrics.

Models. We focus our experiments on the Qwen3 (Yang et al., 2025) family of models as the policy model π_θ. We additionally run some experiments using Gemma 3 and GPT-4o to validate findings. The family of Qwen models is appealing for our understanding of multi-turn scenarios for two reasons: (i) it enables us to study performance over

a range of different model sizes (we focus on 4B - 32B); and (ii) the models are already pre-trained on multi-turn interaction cycles, during the RL stage (Yang et al., 2025), and hence they are well-suited for our analysis. We report all findings by running inference in non-thinking mode (but with reasoning traces using chain-of-thought prompting) and recommended sampling temperature of 0.7. We also ran preliminary experiments with thinking mode, but we found that token limits reduce performance. We use the ReAct (Yao et al., 2023b) framework to perform the roll-outs by interleaving reasoning traces with actions. The number of turns for which the roll-outs are performed is example dependent. Many of our experiments are long-horizon and exceed the context length (involving contexts >32K). In such scenarios, we report results using YaRN (Peng et al., 2023) encoding, following the official recommendation.

Prompting. Across all environments, we engineer environment-specific prompts to ensure the best success rate of the pre-trained models. This helps us ensure that at evaluation time, errors can be attributed with high confidence to limitations of the model rather than prompt construction. Specifically, we: (a) use in-context example trajectories, which demonstrate task-specific reasoning and actions; (b) ensure the in-context examples reflect the information augmented by the oracles during roll-outs. We found the latter to be especially important, as models underperformed if the format of in-context examples was not consistent with environmental feedback during roll-out.

Evaluation Data We perform our analysis by generating multiple unique examples per complexity level per environment. Specifically, for ListWorld, we use 3000 samples with 300 tasks for each of 10 horizon lengths (number of pop operations) ranging from 2-20. For TreeWorld, we use 2400 samples with 300 unique tasks per 8 horizon lengths (number of nodes in tree) configured from 5-60. For GridWorld, we use 1300 samples with 100 unique tasks for each of 13 horizon lengths (length of optimal path) configure to range from 1-60.

Evaluation Metrics. Our primary evaluation metric is the *success rate*, i.e., whether the model completes the task within the specified horizon budget. In addition, we also report the *step accuracy* i.e., whether the agent's action at a particular step in a multi-turn trajectory is optimal. An action is considered optimal, if it is aligned with at least one optimal policy (of multiple non-unique policies) at the current state.

4.3 Setup: Oracle Interventions

To better understand the impact of state tracking, planning, and growing prompt length of multi-turn environments, we conduct a thorough evaluation of the models across all sizes in the presence or absence of our oracle interventions. Each intervention removes one of the aforementioned challenges, and their impact on the agent's performance allows us to understand the importance of each one. None of the oracles use privileged information (i.e., they use the same knowledge available to the LLM). Furthermore, we recognize that the oracle interventions are not necessarily orthogonal to one another and we ablate with all combinations of oracles.

In all environments, the oracle interventions are designed either to *augment* the information provided or *truncate* to sufficient information necessary to achieve the goal. Planning and state tracking interventions augment the context and can be activated or deactivated independently. In contrast, history pruning requires the presence of the state tracking intervention, as it removes all preceding steps from the context, making it impossible for the agent to track the state on its own. In the cases where state tracking intervention is active, the provided state contains sufficient information for optimal decision-making. We provide more details on the oracles in appendix (Section B).

4.4 Results and Analysis

We examine the performance of Qwen3 models on our environments. Fig. 3 shows the success rate of the Qwen3 8B and 32B models on our three environments as a function of the task horizon. We observe that despite the strong performance on problems with a short horizon, the success rate drops drastically as we increase the horizon. This is aligned with the notoriously challenging nature of long-horizon tasks. We study the driving forces behind this phenomenon with our framework.

Overall Oracle Impact. Fig. 4 provides the success rate of 4B, 8B, 14B, and 32B models in each environment for all six oracle configurations, averaged over complexity levels. The addition of each intervention improves the success rate in most cases. The exact impact of each intervention varies among the environments and model sizes.

Model Scale Shifts What Matters. Larger mod-

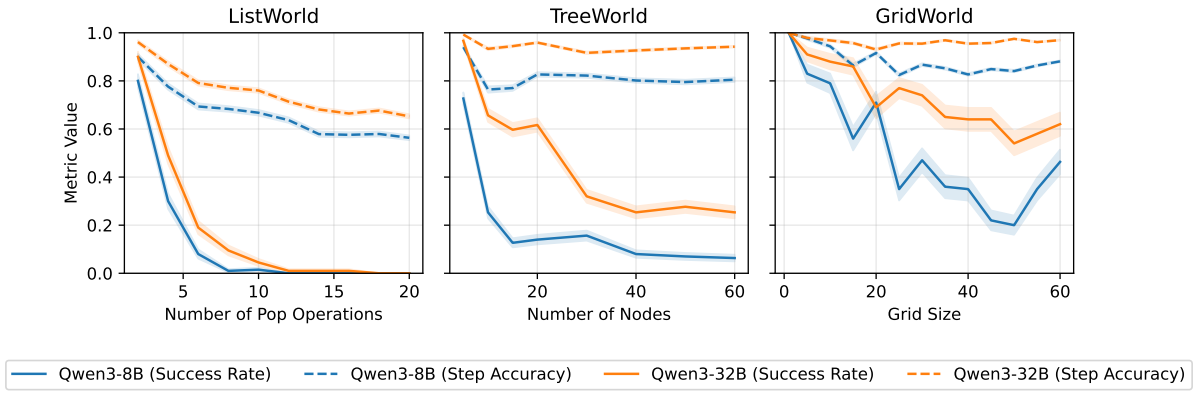


Figure 3: Success rate and step accuracy of Qwen3-8B and Qwen3-32B models by task horizon in ListWorld *(left)*, TreeWorld *(middle)*, and GridWorld *(right)*. Shaded region indicates standard error.

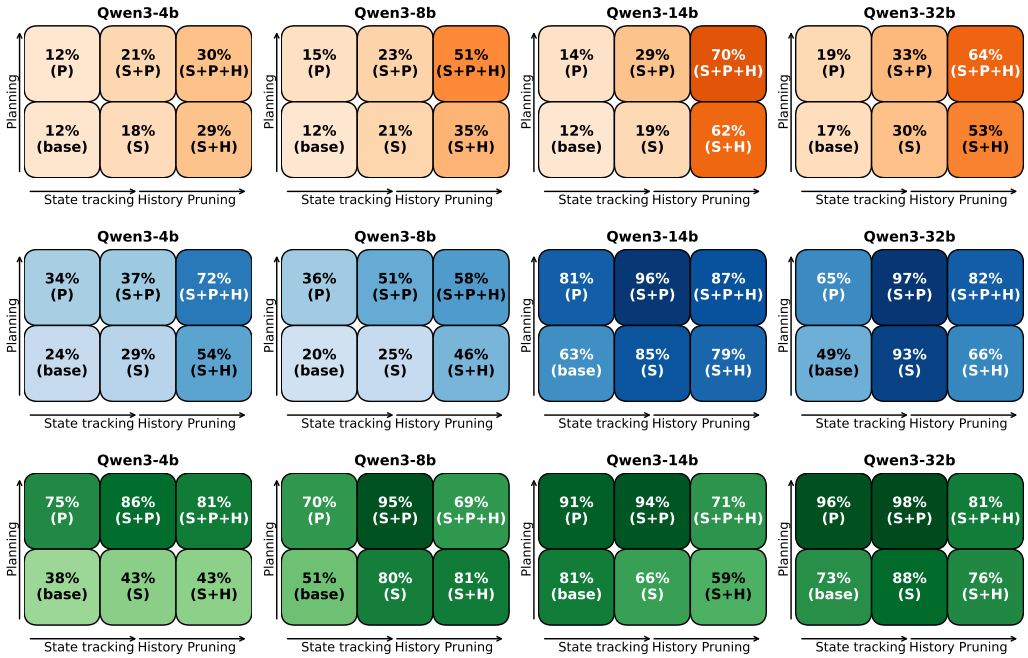


Figure 4: Influence of oracle interventions on ListWorld *(top)*, TreeWorld *(middle)*, and GridWorld *(bottom)*. Results are averaged over all horizon lengths. The labels indicate the active oracles (S: state tracking, P: planning, and H: history pruning).

els outperform smaller models in every environment, and consequently, exhibit less potential for improvement through interventions. To enable insightful comparison of the impact of oracle interventions across model sizes, we adjust for this natural imbalance by picking a longer horizon for larger models and effectively leveling the success rates. For ListWorld and TreeWorld, we choose settings where models succeed about 30% of the time. In GridWorld, larger models never reach this low success rate, and we pick the complexity such that success rate becomes about 75% for all model sizes. Figure 6a shows the impact of the interventions on the success rate for each model size after the leveling selection. The results are averaged over the choice of environment and other interventions. The impact of planning intervention and the difference between task success rate and step accuracy seem generally unaffected by the model size. History pruning is effective for the 4B and 8B models, while the 14B and 32B even suffer from this removal. Another observation is that larger models tend to benefit more from state tracking. This indicates that improving these capabilities in models will have largerly different impact at different model scales.

Rules Dictate the Challenges. Figure 6b compares the impact of each intervention in different environments. Again, we choose the task complexity to level the success rates and average over the choice of model size and other interventions. The oracles' impact significantly varies among environ-

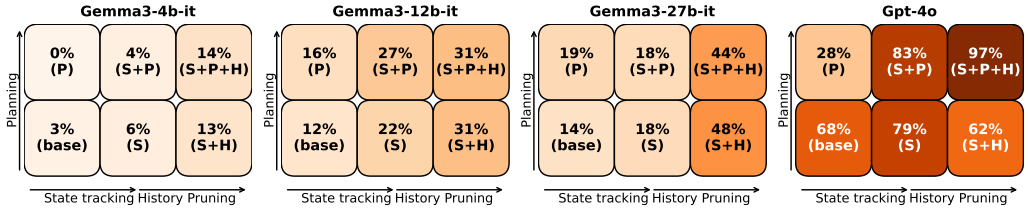
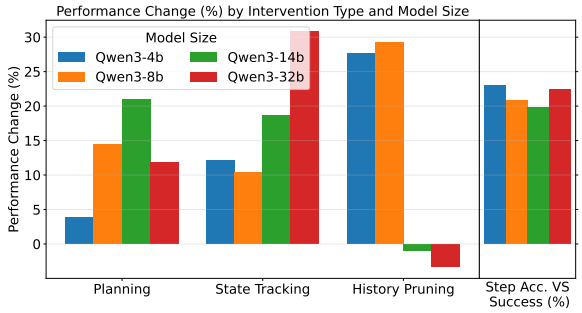
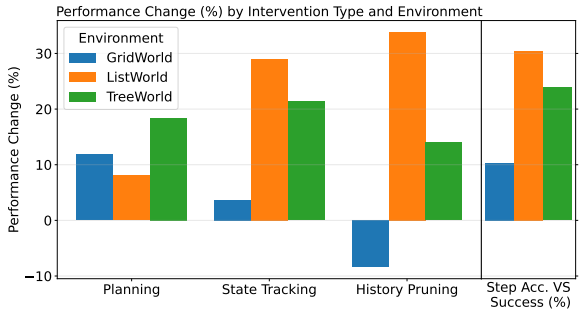


Figure 5: Influence of oracle interventions for Gemma 3 and GPT-4o on ListWorld. Results are averaged over all horizon lengths. The labels indicate the active oracles (S: state tracking, P: planning, and H: history pruning).



(a) Model Size



(b) Environment

Figure 6: Performance change from interventions. The impact of each intervention and varies depending on the model size (a) and environment (b). The relationship between step accuracy and success (%) also varies.

ments. ListWorld performance is boosted best by history pruning, indicating that it is challenging for the agent to attend to the relevant information. Given GridWorld's navigational nature, knowing where to go and also knowing where one comes from is intuitively useful information. This may explain why planning improves while history pruning deteriorates GridWorld performance. While the state tracking oracle has little impact in GridWorld, it is the most impactful oracle in TreeWorld, likely due to complex backtracking required for the task.

Every Step Counts. Our experiments reinforce the hypothesis that the main challenge of multi-turn environments is that success requires *consistently* taking correct actions over many steps. The dashed lines in Fig. 3 show the step accuracy of the model for each value of task horizon. The step accuracy is larger than the task success rate, and even in cases where the success rate is almost zero, a step accuracy above 60% indicates that the majority of the steps are correct. However, as the number of required steps increases, the agent becomes more likely to fail on the task due to the occasional wrong actions it takes. In order to solve these tasks reliably, the agent needs to be almost perfect at all steps. In fact, environments like ListWorld are irrecoverable and a single step leads to failure. The large discrepancy between step accuracy and success rate in Fig. 6b is likely a result of that environment property.

Results with Different Models. In Fig. 5, we provide the success rate of GPT-4o (Hurst et al., 2024) and Gemma 3 models (Team et al., 2025) on ListWorld for different oracle configurations. Gemma 3 results are aligned with the findings from the Qwen3 models. Planning intervention shows minimal benefits while state tracking and history pruning significantly help the model. Both Qwen3 and Gemma 3 models across all sizes benefit from history pruning in ListWorld, but GPT-4o, which is much larger, starts to show signs of performance degradation with history pruning. This further confirms our finding that history pruning benefits smaller models but can hurt larger models.

4.5 Take-aways

To better answer how a practitioner can improve the performance of LLM-based agents on long-horizon tasks, we distill our analysis to actionable guidelines. *First*, improve local action reliability, to prevent terminal mistakes (e.g., ListWorld) and accumulating errors over turns (see Fig. 3). *Second*, tailor solutions that address specific challenges imposed by the environment (e.g., addressing state tracking yielded most gains in ListWorld; Fig. 6b). *Third*, be mindful that success rate gains are influenced by the model size (e.g., pruning context history was more effective in smaller models; Fig. 6a). Taken together, our findings suggest a practical recipe for improving multi-turn performance: catch errors and improve error recovery, identify environment-specific challenges, and account for

the impact of model size.

5 Conclusion

In this paper, we worked towards understanding the discrepancy between LLMs excelling at complex single-turn reasoning tasks and underperforming in multi-turn closed-loop feedback-driven tasks. Key to our approach was to analyze the discrepancy in terms of skills that are required in agent-specific use cases, such as planning, tracking state, and history pruning. To this end, we proposed a counterfactual framework with oracle interventions, where the oracle augments and prunes the information exposed to the agent at each turn. To enable oracle interventions, we additionally propose three procedurally-generated environments, which lets us control task complexity, and more importantly, we can at any turn estimate the set of optimal actions that can be used to guide the agent. Our findings indicate that while oracle skills enable the LLM policy to generally improve, the effectiveness of each skill is also significantly influenced by the model size and the environment. Further, our experiments highlight the importance of close to perfect step accuracy to avoid compounding errors. These insights may guide future development to robust LLM agents for multi-turn tasks.

Limitations

This paper presents the first step towards explaining the performance degradation of capable LLMs in multi-turn long-horizon agent tasks. While we find valuable insights, many important steps remain to fully understand the performance degradation. First, we rely on prompt-based mechanisms to elicit agent-like behavior. While such mechanisms have been shown to be a strong baseline, performance are also influenced by the prompt itself and it is unlikely to find one prompt construction that is optimal for all models, sizes, and environments. This may potentially be alleviated by post-training given specific prompts. Second, we introduce oracle information by augmenting or rewriting the instructions in context visible to the LLM, and hence rely on the LLM to accurately follow the instructions. Although LLMs generally show remarkable instruction following capabilities, it is nonetheless imperfect and a framework to additionally enable an 'instruction following' oracle would strengthen the analysis. Third, we run our analysis on simple programmable environments and benefit from being able to accurately and efficiently construct and isolate oracle interventions. While it is insightful to study this on real-world applications, annotating oracle interventions is often intractable or ill-defined. We do not anticipate immediate risks arising from our work, but we do acknowledge that autonomous agentic system may pose a big threat if faulty, unreliable, or used maliciously. Therefore, we believe that analyses, such as ours, are crucial to mitigate such risks in the future.

References

Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yuexiang Zhai, Kelvin Xu, and Sergey Levine. 2023. LMRL gym: Benchmarks for multi-turn reinforcement learning with language models. *arXiv preprint arXiv:2311.18232*. pages 2

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*. pages 1

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*. pages 1

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI*. pages 2

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, and 1 others. 2025. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*. pages 2

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

MohammadReza Ebrahimi, Sunny Panchal, and Roland Memisevic. 2024. Your context is not an array: Unveiling random access limitations in transformers. *arXiv preprint arXiv:2408.05506*. pages 2

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd

of models. *arXiv preprint arXiv:2407.21783*. pages 1

Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. 2025. Textarena. *arXiv preprint arXiv:2504.11442*. pages 1

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*. pages 1

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *EMNLP*. pages 2

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR. pages 2

Ziheng Huang, Sebastian Gutierrez, Hemanth Kamana, and Stephen MacNeil. 2023. Memory sandbox: Transparent and interactive memory management for conversational agents. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–3. pages 2

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*. pages 8

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*. pages 1

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. 2025. Llms get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*. pages 3

Yubo Li, Xiaobin Shen, Xinyu Yao, Xueying Ding, Yidi Miao, Ramayya Krishnan, and Rema Padman. 2025. Beyond single-turn: A survey on multi-turn interactions with large language models. *arXiv preprint arXiv:2504.04717*. pages 2

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*. pages 1

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. 2025. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *ICML*. pages 1

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565. pages 2

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*. pages 6

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*. pages 2

Stuart Russell, Peter Norvig, and Artificial Intelligence. 1995. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25(27):79–80. pages 2

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. 2025. The illusion of diminishing returns: Measuring long horizon execution in llms. *arXiv preprint arXiv:2509.09677*. pages 2

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2998–3009. pages 2

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*. pages 8

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*. pages 1

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *ACL*. pages 1, 2, 4

Kiran Vodrahalli, Santiago Ontanon, Nilesh Tripuraneni, Kelvin Xu, Sanil Jain, Rakesh Shivanna, Jeffrey Hui, Nishanth Dikkala, Mehran Kazemi, Bahare Fatemi, and 1 others. 2024. Michelangelo: Long context evaluations beyond haystacks via latent structure queries. *arXiv preprint arXiv:2409.12640*. pages 2, 3, 5

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*. pages 2

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023b. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*. pages 2

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, and 1 others. 2024. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 4. pages 1

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*. pages 5, 6

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*. pages 1

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822. pages 2

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*. pages 2, 3, 6

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2025. Generative verifiers: Reward modeling as next-token prediction. In *ICLR*. pages 2

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623. pages 2

Use of AI Assistant

We used approved AI assistants during the course of research to mildly assist with coding (e.g., implementing shortest path algorithm in grid world) and polish writing. In all cases, edits were manually reviewed by authors.

A Prompts

A.1 Prompt: ListWorld

We used the following prompt for ListWorld experiments

````
You are an assistant designed to **modify
lists** through a sequence of simple commands
that you can execute at every turn. You will
be given an initial list and a target list.
Your task is to modify the initial list to a
target list using the following functions,
provided in JSON format:
```json
{
    "pop": {
        "name": "pop",
        "type": "function",
        "description": "Removes an element from
         the environment's list. Index of the
        elements after the removed one will be
        reduced by 1.",
        "parameters": {
            "type": "object",
            "properties": {
                "id": {
                    "type": "integer",
                    "description": "The ID of
                    the element to remove."
                }
            },
            "required": [
                "id"
            ]
        }
    },
    "done": {
        "name": "done",
        "type": "function",
        "description": "Terminates the task.
        Should be called when no more
        operations are needed.",
        "parameters": {
            "type": "object",
            "properties": {}
        }
    }
}
```
The initial list has all the elements of the
target list in the same order, but contains
some extra elements that need to be removed.
To remove them, you should use the 'pop'
function with the index of the element you
want to remove. Remember that once an element
is removed, the index of elements after it
will decrease by 1. The most important rule is
 that you can **only pop the elements from
left to right**. Once you pop an element, the
elements before it (thoses with a smaller
index) can no longer be removed, and you will
get an error if you try to do so. Once you are
 done and have turned the initial list to the
target list, you should call the 'done'
function.

You must reason step-by-step, choosing actions
 based on the current state of the list. Avoid
 redundant queries and aim for efficiency.
Provide your response as a single python
function call enclosed in a code block:
```python
function_call(arg1=val1, arg2=val2, ...)
```

=== Starting new task ===
Initial list: ${initial_list}
Target list: ${target_list}
Your task is to modify the initial list to
target list by calling the 'pop' function.
Call 'done' function once you are done.
Remember, **only pop the elements from left to
 right**.
````

A.2 Prompt: TreeWorld

We used the following prompt for tree world:

````
SYSTEM_PROMPT = f"""You are a reasoning agent
 searching a tree for a node with a specific
value (which may or may not be reachable by
you). Each node has two attributes: (1) "id" (
unique string) and (2) "value" (unique integer
). In each task, you are provided with a
partial information about some of the nodes in
 the tree. For each node included in this
information, you are given the id and the
value. For some of the nodes the list of the
node's children ids is also provided. The
format in that case is: (id=<node_id>, value=<
node_value>) -> [<child_id1>, <child_id2>,
...]. Note that an empty list indicates that
the node is a leaf and has no children. For
other nodes, the children are not given to you.
 In that case, you are given: \"UNKNOWN\" in
place of the children ids.

A target value will be given to you at the
beginning of each task. Your job is to try to
find a node with this value and report its id.
 You should do this using the following
functions, provided in JSON format:
```json
{
    "get_children": {
        "name": "get_children",
        "type": "function",
        "description": "Returns the list of
        children nodes for a given node ID (i.e
        ., the outgoing edges)",
        "parameters": {
            "type": "object",
            "properties": {
                "id": {
                    "type": "string",
                    "description": "The ID of
                    the node whose children are
                    to be retrieved."
                }
            },
````
````

```
            "required": [
                "id"
            ]
        },
        "returns": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "id": {
                        "type": "string"
                    },
                    "val": {
                        "type": "integer"
                    }
                },
                "required": [
                    "id",
                    "val"
                ]
            },
            "description": "List of child nodes
             as objects with 'id' and 'val'."
        }
    },
    "found": {
        "name": "found",
        "type": "function",
        "description": "Indicates that the node
         with the specified ID contains the
        target value.",
        "parameters": {
            "type": "object",
            "properties": {
                "id": {
                    "type": "string",
                    "description": "The ID of
                    the node that contains the
                    target value."
                }
            },
            "required": [
                "id"
            ]
        },
        "returns": {
            "type": "string",
            "description": "Confirmation that
            the node with the given ID contains
             the target value."
        }
    },
    "unreachable": {
        "name": "unreachable",
        "type": "function",
        "description": "Indicates that the node
         with the target value is impossible to
         reach.",
        "parameters": {
            "type": "object",
            "properties": {}
        },
        "returns": {
            "type": "string",
            "description": "Confirmation that
            the target value was not possible
            to find in the tree."
        }
    }
}
```

```
‘‘‘
You can ask for the ids and values of the
children of a node by calling the ‘
get_children‘ function with the node's id.
If you find the target node (the node with the
 target value), return its id using the ‘found‘
 function. After calling ‘found‘ the task will
 terminate and you succeed if you have
reported the correct id.
If you believe it is impossible to find the
target node, call the ‘unreachable‘ function.
After calling ‘unreachable‘ the task will
terminate and you succeed if it was impossible
 for you to find the target node.

You must reason step-by-step, choosing actions
 based on the current state of the search.
Avoid redundant queries and aim for efficiency.

Provide your response as a single python
function call enclosed in a code block:
‘‘‘python
function_call(arg1=val1, arg2=val2, ...)
‘‘‘

=== Starting new task ===
Your task is as follows: Find the id of the
node with value **${target_node_val}**
Once you find the target node containing this
value, return its id by calling ‘found‘
function. If you think it is impossible to
find this node, call the ‘unreachable‘
function.
Provide all responses as a single python
function call enclosed in a code block.
```

A.3 Prompt: GridWorld

We used the prompt below for GridWorld:

```
You are an intelligent agent playing a grid
world navigation game. Your goal is to move
from the given start position to the goal
position using the fewest possible moves. The
game board is a 2D grid with the following
properties:

- The top-left corner is coordinate (0, 0),
and the bottom-right corner is (size-1, size
-1).
- You will be given:
    * The size of the board (N x N)
    * Your starting position (row_index,
    column_index)
    * The goal position (row_index,
    column_index)
    * A list of hole positions (each a
    coordinate)
    * The maximum number of moves allowed
- You can move using these actions: ‘up()‘, ‘
down()‘, ‘left()‘, ‘right()‘
- *Only* if you have reached the goal, call ‘
done()‘ to terminate the game. Once you
terminate the game, you are not allowed any
more moves.
- You can reason, but always end by specifying
 a single action within triple fenced blocks.
Example
‘‘‘python
up()
```

```
```
or
```python
done()
```
- Each move costs **1 move**.
- If you move into a hole, you incur a **
penalty of 3 additional moves** (because it is
 hard to get out of a hole).
- You must stay within the grid boundaries.
- Your objective: **Reach the goal in as few
moves as possible without exceeding the
maximum allowed moves.**
- After each move, you will receive the
updated position and remaining moves.
- In the triple fenced blocks, do not write
anything except the next action in the
required format.

=== Your Task ===
The grid world game is set up as follows:
- Board size: ${size} x ${size}
- Start position: ${start}
- Goal position: ${goal}
- Holes at: ${holes}
- Your move budget is: ${max_moves}

Your task: Navigate from the start to the goal
 using the fewest moves possible. Remember:
- You can move using the following actions: `
up()`, `down()`, `left()`, `right()`
- If you reached the goal, terminate by
performing action `done()`
- Each action must be in a triple-fenced
Python code block, like:
```python
right()
```
- Avoid holes if possible, as they cost extra
moves.
- Do not exceed the maximum allowed moves.

Begin your first move now.
"""
```

## B Implementation Details: Oracle Interventions

In this section, we elaborate on the implementation details of the oracles used in the paper.

**Common details.** In all the environments, the oracle interventions are designed either to *augment* the information provided or *truncate* to sufficient information necessary to achieve the goal. All augmentations and truncations by the oracle are performed by modifying the context visible to the LLM. Furthermore, our oracles do not use privileged information and hence operate using the same information available to the LLM.

**Planning.** The planning oracle appends the LLM's context with a short description of a one-step subtask, such that the model can bypass reasoning about future steps. In fully observable environments (e.g., in ListWorld, GridWorld), the oracle appends a hint on the next state to reach (e.g., "you should first reach (3, 4)" in GridWorld). In partially-observable environments (e.g., TreeWorld), the oracle appends a hint on whether the continue exploring (e.g., "ask for children of a node that you haven't asked yet") or leveraging existing information (e.g., "report the id of the node you found").

**State Tracking.** Given the knowledge of the current multi-turn interaction visible to the LLM, the state tracking oracle at each step provides a brief summary of the state. In some cases, this can be succinct (e.g., mapping initial location and subsequent actions of the agent into a hint "your current location: (3, 4)"), whereas in other cases it can be verbose due to the agent acquiring new information (e.g., summary of nodes and edges in TreeWorld).

**History Pruning.** In the case where the state tracking oracle is active, we can further leverage the history pruning oracle to accurately compress the context. We implement this by: (a) recovering the exact state of the agent using the state tracking oracle, (b) removing all preceding user-assistant steps from the context, and (c) finally re-writing the task description to account for the current state. For instance, in GridWorld (see Fig. 1c), we rewrite the task description by substitute the starting location of the agent with the current location.
```