Learning and Adaptation in Interactive AI Agents

by

Yao Fu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2026

Doctoral Committee:

Professor Honglak Lee, Chair
Professor Satinder Singh Baveja
Professor Bernadette Bucher
Professor Joyce Chai

Yao Fu
violetfy@umich.edu
ORCID iD: 0000-0001-6046-9896

© Yao Fu 2026

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Professor Honglak Lee, for his invaluable insights and mentorship throughout my doctoral research. I am also sincerely grateful to my dissertation committee members, Professor Satinder Singh Baveja, Professor Bernadette Bucher, and Professor Joyce Chai, for their time, thoughtful feedback, and valuable advice on my research and dissertation.

I am deeply thankful to all my collaborators, whose ideas and discussions have contributed greatly to my research during my PhD: Kyunghoon Bae, Jongwook Choi, Pierluca D'Oro, Yijie Guo, Dong-Ki Kim, Jaekyeom Kim, Moontae Lee, Anthony Z. Liu, Lajanugen Logeswaran, Run Peng, Sathvika Ayyappa Prabhu, Shengyi Qian, Ran Qiu, Jacob Sansom, Sungryull Sohn, Huijie Tang, Joseph Tighe, Manchen Wang, Xinhe Wang, and Fanyi Xiao.

Finally, I would like to thank my family and my boyfriend for their unconditional love and support. They have been my source of strength throughout this journey.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

LIST OF FIGURES vi

LIST OF TABLES x

LIST OF APPENDICES xiii

ABSTRACT xiv

CHAPTER

1 Introduction 1

1.1 Summary of Research Contributions 3
1.1.1 Automated Generation and Selection of Context-Aware Guidelines for Large Language Model Agents (Chapter 2) 3
1.1.2 Beyond Blind Following: Evaluating Robustness of LLM Agents under Imperfect Guidance (Chapter 3) 4
1.1.3 Go Beyond Imagination: Maximizing Episodic Reachability with World Models (Chapter 4) 4
1.1.4 Hierarchical Procedural Meta-Reasoning for Generalizable Multimodal Agents (Chapter 5) 5
1.1.5 Evidence-Grounded Verification for Interactive LLM Agents (Chapter 6) 6

2 Automated Generation and Selection of Context-Aware Guidelines for Large Language Model Agents 7

2.1 Introduction 7
2.2 Related Work 9
2.3 AutoGuide: Principled Method Based on Context-Aware Guidelines 10
2.3.1 Problem Statement 10
2.3.2 Extraction of Context-Aware Guidelines 11
2.3.3 Applying Context-Aware Guidelines at Test Time 12
2.4 Evaluation 13
2.4.1 Evaluation Setup 13
2.4.2 Main Results 16
2.4.3 Analyses of AutoGuide 19

2.5 Conclusion . . . 20

3 Beyond Blind Following: Evaluating Robustness of LLM Agents under Imperfect Guidance . . . 21

3.1 Introduction . . . 21
3.2 Related Work . . . 24
3.3 MIRAGE Benchmark Framework . . . 25
3.3.1 Problem Setting . . . 25
3.3.2 Benchmark Overview . . . 25
3.3.3 Guidance Imperfection . . . 26
3.3.4 Environment Implementation and Task Construction . . . 27
3.3.5 Extensibility and Difficulty Scaling . . . 28
3.4 Experiments . . . 28
3.4.1 MIRAGE-World Results . . . 29
3.4.2 MIRAGE-Web Results . . . 31
3.4.3 Additional Analyses . . . 31
3.5 Conclusion . . . 34

4 Go Beyond Imagination: Maximizing Episodic Reachability with World Models . . . 35

4.1 Introduction . . . 35
4.2 Method . . . 37
4.2.1 Go Beyond Imagination . . . 37
4.2.2 Conceptual Advantage of GoBI over Prior Works . . . 40
4.3 Experiments . . . 41
4.3.1 MiniGrid Navigation Tasks . . . 41
4.3.2 Experiments on Control Tasks . . . 44
4.3.3 Ablation Study . . . 45
4.4 Related Work . . . 47
4.4.1 Exploration in Reinforcement Learning . . . 47
4.4.2 Episodic Memory . . . 48
4.4.3 Learning World Models with Forward Dynamics . . . 48
4.5 Discussion: Test-Time Exploration of LLM Agents . . . 49
4.5.1 Method . . . 50
4.5.2 Preliminary Study . . . 50
4.6 Conclusion . . . 51

5 Hierarchical Procedural Meta-Reasoning for Generalizable Multimodal Agents . . . 52

5.1 Introduction . . . 52
5.2 Related Work . . . 54
5.3 Method . . . 56
5.3.1 Background . . . 56
5.3.2 Procedure-Aware Multimodal Agent with Meta-Reasoning . . . 56
5.3.3 Procedure-Aware Grounded Executor . . . 57

5.3.4 High-Level Procedural Reasoner . . . 58
5.4 Experiments . . . 60
5.4.1 Experimental Setup . . . 60
5.4.2 Cross-Task and Cross-Application Generalization . . . 62
5.4.3 Application Version Update Adaptation . . . 63
5.4.4 Compositional Multi-App Generalization . . . 64
5.4.5 Ablation . . . 65
5.5 Conclusion . . . 67
6 Against Epistemic Shortcuts: Evidence-Grounded Verification for LLM Agents . . . 68
6.1 Introduction . . . 68
6.2 Related Work . . . 70
6.3 Method . . . 71
6.3.1 Problem Setting . . . 71
6.3.2 Evidence-Grounded Verification for Interactive LLM Agents . . . 72
6.4 Evaluation . . . 74
6.4.1 Experimental Setup . . . 74
6.4.2 Training with Monitor-Derived Supervision . . . 75
6.4.3 Test-Time Epistemic Monitoring . . . 75
6.5 Conclusion . . . 76
7 Conclusion and Future Work . . . 78
7.1 Key Findings and Contributions . . . 78
7.2 Future Work . . . 79

APPENDICES . . . 81

BIBLIOGRAPHY . . . 149

LIST OF FIGURES

FIGURE

2.1 AutoGuide aims to extract the implicit knowledge embedded in offline experiences and help the decision-making process of an LLM agent. Specifically, our method generates a comprehensive set of context-aware guidelines from offline data and explicitly identifies when each guideline is applicable by generating its corresponding context. Our context-aware guidelines enable providing pertinent guidelines at test time by identifying the context of the current trajectory, leading to correct decision-making compared to baselines without context-aware guidelines. 8

2.2 Context-aware guideline generation process based on a pair of contrastive trajectories τ_+^i and τ_-^i. In this example, the two trajectories start deviating from each other at $t=0$. The context identification module generates a description of the context at $t=0$ given $\tau_{:0}^i$, and the guideline extraction module generates the corresponding guideline for that context. 10

2.3 Sequential decision-making benchmark domains considered in our work: ALFWorld [135], WebShop [169], WebArena [188], and multimodal real-world websites. Graphic credit: [134, 169, 188, 1]. 14

2.4 Trajectories of ReAct, ExpeL, and AutoGuide from the same test task. ReAct (Left) chose the wrong item, consequently failing the task in the end. ExpeL (middle) was confused by guidelines that were irrelevant to current context, leading to incorrect reasoning and actions. AutoGuide (right) selects relevant guidelines to the agent's context, enabling the agent to accomplish the task. . . 17

3.1 Examples of four types of guidance imperfection. (1) Incomplete: a critical step (e.g., uploading a photo to review an order) is omitted; (2) Incorrect: a weapon crafting recipe includes wrong components; (3) Misaligned: originally correct navigation guidance ("turn left") becomes invalid because the road is under construction; (4) Underspecified: cooking instructions are vague and high-level, lacking actionable detail. 22

3.2 Overview of MIRAGE. For navigation, cooking, and gaming, we first generate the environment and task instance (e.g., map layout, recipe goal, or target boss). From this, we derive a perfect solution path, which is then optionally perturbed to produce an imperfect guide. For web tasks, we extract guidance directly from human demonstrations that we collected. We introduce two types of perturbations: Env Change (blue), which alters the environment itself, and Guide Change (orange), which corrupts the instruction. 23

3.3 Two examples where the agent fails under imperfect guidance. Left: it does not recognize the misalignment. Right: it realizes the guidance is inapplicable but cannot recover and continues with flawed actions. 30
3.4 Effect of skipping steps in web guidance. 32
3.5 Error types of GPT-4.1-mini ReAct agents. 33

4.1 Illustration of how GoBI works on MiniGrid. For the environment in the upper-left corner, the red triangle indicates the position and orientation of the agent. It has a 7×7 partially-observable view (highlighted). During pre-training (Stage 1), we collect data using a random policy to train a forward dynamics model $\hat{f}_\phi(pano_t, a_t) = o_{t+1}$, where $pano_t$ denotes the panoramic view as defined in Section 4.3.1. For policy training (Stage 2), we apply $\hat{f}$ to predict observations in the future k timesteps with n random actions for each step. We add the new ones to an episodic buffer $\mathcal{M}$ and take the change of size of $\mathcal{M}$ as the episodic intrinsic reward r^{epi}. The lifelong intrinsic reward is COUNT-based. Our intrinsic reward GoBI is $r^{\text{lifelong}} * r^{\text{epi}}$. 36
4.2 An illustration of how our episodic buffer updates. We consider reachable states that are $k = 2$ timesteps away from s_t. After the agent moves from s_1 to s_2, the episodic buffer $\mathcal{M}$ (shaded in green) expands by 3 new reachable states (shaded in yellow). More formally, $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{M}_R(s_2)$, where $\mathcal{M}_R(s_2)$ indicates a set of all the states 2-step reachable from s_2. Notice that all the states in the trajectory, i.e., s_0, s_1, s_2 are also added to the buffer. 38
4.3 Rendering of the environments used in this chapter. Left: 2D grid world navigation tasks that require object interactions. Right: DeepMind Control tasks with visual observations. 41
4.4 Visitation heatmaps on KeyCorridorS5R3 at different training stages. This figure compares the policy behavior of GoBI and NovelD. A dark red color means plentiful visitations, white means the agent has seen the space but did not step on it, and black indicates spaces that are not discovered. It is worth noticing that early in the training (3.5M and 7M timesteps), our policy already learns not to go into an empty room, likely because states in an empty room are easily predictable. On the contrary, even after 11M steps, an agent trained with NovelD still goes into an empty room (bottom-right corner) for more state visitations. . 42
4.5 Training performance of GoBI and the baselines on 12 MiniGrid environments. The x-axis shows the number of environment steps. We shift the training curves towards right by the number of environment steps we use to pre-train the dynamics model, i.e., 1e5 timesteps. Results are averaged across 4 seeds. 43
4.6 Training curves of GoBI and the baselines on DeepMind Control Suite. The curves are averaged across 5 seeds. 45
4.7 Training performance comparison among GoBI, R1, R2, and R3 on 3 MiniGrid environments. 46
4.8 Comparison between using the real dynamics model of the environments vs using a learned one on MiniGrid environments. In both MultiRoom and KeyCorridor, using a real dynamics model to derive intrinsic reward makes the policy converge faster, especially on KeyCorridor. 47

4.9 We make forward predictions for different numbers of future steps k using both the real dynamics and the learned dynamics model. The plots above show the training performance on MiniGrid MultiRoom-N7-S8. 48

5.1 Overview of Procedure-Aware Multimodal Agent with Meta-Reasoning (PAMAR). 53
5.2 Examples of application version updates in DigiData-AppUpdate. Left: An Airbnb update removes category shortcuts below the search bar (e.g., *Top of the world*), breaking procedures that rely on these entry points. Right: A calculator redesign changes layout and visual structure while preserving core functionality, illustrating workflow/UI drift. 60
5.3 **Ablation Study: Training–Test Procedure Alignment**. Success rates (%) on DigiData-Bench-Auto for three evaluation settings: no procedure (None), static procedure (Static) and dynamic procedure (Dynamic). All procedures are generated with GPT-4o. 66

6.1 Illustration of how EVIA works. In this example, the agent hallucinates that Brand B is gluten-free based on its internal knowledge, without specifically checking whether this bagel is gluten-free. If it is, this leads to a successful trajectory with a shortcut; otherwise, it directly causes the trajectory to fail. The epistemic monitor detects that the action *Buy Brand B* is not supported by sufficient environmental evidence, rejects the action, and asks the agent to regenerate. This either works directly at test time to prevent shortcut behavior, or naturally constructs preference pairs for further fine-tuning. 70

A.1 Our prompt template for context identification (Equation 2.1) in the WebShop domain. 89
A.2 Our prompt template for context identification (Equation 2.1) in the ALFWorld domain. 90
A.3 Our prompt template for context identification (Equation 2.1) in the WebArena domain. 91
A.4 Our prompt template for guideline extraction (Equation 2.2) in the WebShop domain. 92
A.5 Our prompt template for guideline extraction (Equation 2.2) in the ALFWorld domain. 92
A.6 Our prompt template for guideline extraction (Equation 2.2) in the WebArena domain. 93
A.7 Our prompt template for context matching (Section 2.3.3) in all the WebShop, ALFWorld, and WebArena domains. 94
A.8 Our prompt template for selecting the most relevant context-aware guidelines at test time (Equation 2.3 from Section 2.3.3) in the WebShop and ALFWorld domains. 94
A.9 Our prompt template for selecting the most relevant context-aware guidelines at test time (Equation 2.3) in the WebArena domain. 94
A.10 Example contexts and corresponding guidelines for WebArena. 95
A.11 Example contexts and corresponding guidelines for real-world websites. 96

B.1 Four navigation environment variants. From left to right: (1) Standard — typical terrain layout; (2) Seasonal — snowmen and ice blocks appear, flowers and mushrooms disappear, and trees become snow-covered; (3) Construction — an intersection is blocked to simulate roadwork, requiring the agent to find a detour; (4) Dynamic — a pedestrian moves toward the agent, and decorative lamps change color at each timestep. 100
B.2 Example of narrative-style rewrite. Guidance is rewritten in a more descriptive form. 123
B.3 Example of merged rewrite. Multiple consecutive steps are compressed into a single abstract description. 124
B.4 Example of UI label variation. Labels for interface elements are changed to introduce ambiguity or misalignment. 124

C.1 Training performance of GoBI on MiniGrid MultiRoom-N12-S10 with different intrinsic reward decay ρ. 131
C.2 Training performance of GoBI on MiniGrid MultiRoom-N7-S8 with different n randomly sampled actions per step. 132
C.3 Heatmaps of example trajectories at different training steps in MiniGrid-MultiRoom-N7-S8 environment. 133
C.4 Training performance comparison on MultiRoom-N7-S8 environment among 1) use a fixed pre-trained dynamics model, 2) use a pre-trained dynamics model, and fine-tune it online, 3) no pre-training, directly train the dynamics model together with policy training. 133
C.5 An illustrative example of why R2 does not work well to encourage efficient exploration. In this example, our goal is to include all the states into the episodic buffer quickly. GoBI can move directly to the center in one step for maximum intrinsic reward, while R2 may choose to take extra steps for exploration since it mainly focuses on whether the episodic buffer expands or not. 134

LIST OF TABLES

TABLE

2.1 Test reward and success rate on ALFWorld, WebShop, and WebArena. The base agent model for ALFWorld and WebShop is GPT-3.5-turbo and for WebArena is GPT-4-turbo. Reflexion is done by GPT-4-turbo for at most 3 trials. In our experiments, due to token limit of GPT, we did not experiment with Reflexion on WebArena tasks. 16
2.2 Test results of AutoGuide on 3 real-world web domains within multimodal settings. The base agent model runs with GPT-4V and applying context-aware guidelines significantly improves the performance. 18
2.3 Analysis of AutoGuide against ReAct with varying numbers of in-context examples. 18
2.4 Ablation study of AutoGuide using various top-k values. 18
2.5 Out-of-distribution generalization of context-aware guidelines from WebShop on the 98 WebArena–Shopping tasks that have a product in the intent template. . 20
2.6 Ablation study of AutoGuide, analyzing each module's contribution in WebShop. CI denotes our context identification module, and GES denotes the guideline extraction and selection modules. 20

3.1 Overview of MIRAGE, including task statistics and example imperfect guidance. 26
3.2 MIRAGE-World results for ReAct, Plan & Solve (P&S), and Reflexion across different guidance. ReAct and P&S are evaluated for 1 trial and Reflexion for up to 3 trials. Numbers denote success rates (%) out of 100 tasks per condition, averaged across merged categories and rounded to the nearest integer. Abbreviations: **4.1m** = GPT-4.1-mini, **4om** = GPT-4o-mini, **Qwen14** = Qwen-3 14B, **Qwen235** = Qwen-3 235B, **Llama** = LLaMA-3 70B. 29
3.3 ReAct success rate with GPT-4o on WebArena tasks with imperfect guidance. . 31
3.4 Comparison between **NA** = Non-Aware and **IA** = Imperfect-Aware agents. . . 32

4.1 Preliminary comparison of success rates (%) on two MIRAGE-Misaligned Roguelike variants. The backbone model is `Qwen3.5-35B-A3B`. 50

5.1 Results on DigiData-Bench-Auto. The reported numbers are success rate (%) on cross-task, cross-app generalization tasks, and the whole evaluation set. Search(%) denotes the fraction of episodes that invoke external knowledge acquisition. 61

5.2 Results on DigiData-AppUpdate. SR(%) denotes success rate. Search(%) is the fraction of episodes that query external update notes. Procedures are generated with GPT-4o. 64
5.3 DigiData-MultiApp Results. These tasks require navigation across two applications. SR(%) denotes the percentage of successfully completed episodes, while Search(%) denotes the percentage of episodes that require querying external information. 65
5.4 **Ablation: Meta-Reasoner Trigger Timing.** Success rate on DigiData-Bench-Auto (All split). Delaying the first predicted retrieval trigger noticeably degrades performance. 67

6.1 Mean accuracy on LOCA-bench across different context-length settings for ReAct, SFT, Step-DPO, and EVIA-DPO. 75
6.2 Test-time success rates (%) of ReAct and EVIA on PackAudit and WebShop and mean accuracy on LOCA-bench. 76

A.1 Experiment hyper-parameters on ALFWorld. The maximum allowed episode length and n-shots follow the same setup in ReAct. 83
A.2 Experiment hyper-parameters on WebShop. The maximum allowed episode length, the number of search results per page, and n-shots follow the same setup in ReAct. 85
A.3 Experiment hyper-parameters on WebArena. The number of shots follows the same setup in ReAct. 86
A.4 Experiment hyper-parameters on multimodal real-world website tasks. 88

B.1 MIRAGE-World results for ReAct across different guidance conditions. All results are from a single trial per task. Each number denotes the success rate (%) out of 100 tasks (10 tasks across 10 seeds), rounded to the nearest integer. Standard errors are computed across seeds. The column labeled **G.** indicates the type of guidance: **P** = Perfect, **NA** = No guidance, **IN** = Incorrect, **IC** = Incomplete, **M** = Misaligned, **U** = Underspecified. 117
B.2 ReAct setting: success rates (%) under different guidance qualities for smaller models. M = Misaligned, I = Incorrect. 120
B.3 Comparison of ReAct, Plan & Solve, and Reflexion using GPT-4.1-mini across all tasks. Numbers denote success rates (%) averaged over 10 tasks per seed, with 10 different seeds. Reflexion allows 3 trials per task, while the others use only 1. Standard errors are computed across seeds. 121
B.4 Task success rate (%) and standard error for ReAct (GPT-4o) on WebArena tasks under different guidance conditions. Columns are ordered left to right from perfect guidance to imperfect guidance, and then to no guidance. 122
B.5 Performance on ambiguous vs. unambiguous cooking tasks with inquiry action. Abbreviations: **Std-R** = Standard ReAct; **Imp-R** = Imperfect-Aware ReAct; **Ambig. (new)** = ambiguous task with perfect guidance; **Unambig. (original)** = unambiguous task with perfect guidance. 122

C.1 Hyper-parameters for experiments on MiniGrid. These hyper-parameters are shared across all the methods 127
C.2 The hyper-parameters of GoBI for experiments on MiniGrid. 128
C.3 Hyper-parameters for experiments on DeepMind Control Suite. These hyper-parameters are shared across all the methods 130
C.4 Hyper-parameters for experiments on DeepMind Control Suite. These hyper-parameters are specific to our method. 130
C.5 Wallclock training time comparison in hours between NovelD and GoBI. 132

LIST OF APPENDICES

APPENDIX

A Automated Generation and Selection of Context-Aware Guidelines for Large Language Model Agents **81**

B Beyond Blind Following: Evaluating Robustness of LLM Agents under Imperfect Guidance **97**

C Go Beyond Imagination: Maximizing Episodic Reachability with World Models **125**

D Hierarchical Procedural Meta-Reasoning for Generalizable Multimodal Agents **135**

E Against Epistemic Shortcuts: Evidence-Grounded Verification for LLM Agents **147**

ABSTRACT

Recent advances in foundation models have endowed interactive agents with capabilities in language understanding, commonsense reasoning, and multimodal perception. While these capabilities enable agents to operate in complex environments, reliable real-world deployment remains challenging. Agents must not only efficiently acquire task knowledge but also flexibly adapt it when facing unseen, dynamic, or evolving environments. This dissertation studies the effective learning and adaptation of such AI agents.

The first part of this work focuses on learning explicit natural language representations of task-specific knowledge. We introduce AutoGuide, a framework that summarizes desired behaviors into natural language guidelines. These guidelines are stored and retrieved in a context-aware manner to enhance decision-making at test time.

However, whether domain knowledge is explicitly expressed as guidelines or implicitly encoded in model parameters, it can easily become misaligned with the actual environment. The second part systematically examines this limitation by introducing MIRAGE, a benchmark for evaluating the robustness of large language model agents with imperfect guidance. MIRAGE highlights agents' failures to identify, verify, or recover from flawed instructions.

The third part investigates methods to improve agent performance when prior knowledge is insufficient or misleading. We first introduce GoBI, which applies world-model-based intrinsic rewards to encourage efficient exploration in procedurally generated environments. Building on this, we study test-time exploration for LLM-based agents in text-based environments. Furthermore, we propose PAMAR, a hierarchical framework combining high-level procedure generation, action grounding, and selective external tool use to improve mobile navigation in settings with diverse applications and frequent updates. Finally, we investigate another common failure mode: shortcut behaviors caused by hallucinations or by spurious action patterns learned from imperfect training data. To mitigate this, we propose EVIA, which incorporates an LLM-based information-sufficiency verifier. This verifier can serve as either a test-time safeguard or a training-time mechanism for improving data quality.

Ultimately, this dissertation advances the development of interactive agents through more sample-efficient agent learning and more reliable adaptation under uncertainty and distribution shifts inherent to real-world deployment.

CHAPTER 1

Introduction

Interactive agents are AI systems that generate actions to interact with environments. In recent years, agents have attracted growing interest from both practical and scientific perspectives. From a practical perspective, agents have the potential to automate complex real-world tasks, such as software development [180, 57, 168], web navigation [22, 50, 185], and online shopping [169, 150, 90]. From a scientific perspective, interactive agents can also help us better understand intelligence, especially how behaviors emerge through interaction.

Large language models (LLMs) have substantially expanded the potential of interactive agents. In traditional agent designs, most agents were developed for specific tasks and environments [144, 125, 128]. They typically operated within predefined observation and action spaces [10], with task knowledge encoded in learned policies, task-specific rules, or symbolic representations. Adapting such agents to new tasks often required additional fine-tuning or system redesign. LLMs relax many of these constraints. Their ability to generate natural language and process multimodal inputs enables them to operate across a much broader range of tasks [162, 170, 185, 148, 161]. Their broad prior knowledge and reasoning capabilities also enable them to critique, verify, and refine actions, leading to higher-quality data and improved decision-making [132, 92, 38, 67].

Despite these advances, learning task knowledge in a sample-efficient manner and operating reliably in dynamic, unseen, or continuously evolving environments remain open challenges for interactive agents. In light of these challenges, this dissertation investigates the following question: *How can agents acquire, represent, and adapt task knowledge for reliable decision-making?* We approach this question from three complementary perspectives: knowledge representation, agent robustness, and effective adaptation.

First, we investigate how task knowledge can be represented and retrieved efficiently. In traditional reinforcement learning, agents learn from successful and failed trajectories by updating model weights [128, 45]; however, this process is highly sample-inefficient for LLMs. Alternatively, by comparing these trajectories, we can attempt to explicitly extract what

actions lead to success and what mistakes to avoid, translating them into interpretable, transferable, and easily revisable natural language guidelines [184, 41]. However, an agent operating across diverse situations may accumulate an extensive body of knowledge. Providing all accumulated knowledge is inefficient, and retrieving irrelevant knowledge may distract the agent from effective decision-making. The central challenge is therefore twofold: how to formulate context-specific knowledge in natural-language form, and how to retrieve the knowledge most relevant to the agent's current context. To address these challenges, we develop AutoGuide, which derives context-specific natural-language guidelines from offline data and indexes them by context for efficient retrieval.

Second, we investigate the vulnerability of agents when this explicit knowledge is flawed. While relying on natural language guidelines or retrieving external auxiliary knowledge supports better decision-making [60, 58], such knowledge is not guaranteed to be perfect [149, 175, 102]. LLM-generated guidelines may contain hallucinations, human-written manuals may include omissions or errors, and online tutorials may become outdated over time. Therefore, it is important to understand agents' failure modes when the retrieved knowledge is mismatched with the test-time environment. This is particularly important for agents based on instruction-tuned LLMs, whose decisions are strongly influenced by the provided context. We therefore introduce MIRAGE, a benchmark for systematically evaluating the robustness of LLM agents under incorrect, incomplete, misaligned, or underspecified guidance.

Third, we investigate how to teach agents to adapt more effectively at test time. We consider this problem at three levels.

At the *objective level*, when the agent lacks sufficient knowledge to make effective progress toward the task goal, exploration can serve as an auxiliary objective for acquiring the necessary knowledge. Exploration has been widely studied in reinforcement learning [120, 108, 181]. We therefore revisit the intrinsic-motivation principles in GoBI, which maximizes episodic reachability in procedurally generated environments to enable effective exploration in unseen layouts. We then investigate how these principles can inform the design of effective test-time exploration strategies for LLM-based agents.

At the *architecture level*, adaptation difficulty can be reduced through functional decomposition, decoupling high-level planning from low-level action grounding. LLMs possess substantial prior knowledge for generating and updating procedural plans [26, 55], as well as refining them using external tools [127] such as online search. Through this decomposition, we simplify agent learning by shifting the problem from learning a direct mapping from observations to actions to grounding natural-language procedural steps to the environment. Building on this, we introduce PAMAR, which enables agents to learn effective

action grounding while allowing LLMs to selectively retrieve external information only when necessary.

Finally, at the *criterion level*, effective test-time adaptation to similar cases requires more than blindly imitating action patterns; agents must understand the rationale behind actions to learn the principles that govern action selection. Prior outcome-based mechanisms for selecting demonstrations or in-context learning examples overlook the fact that a *successful* trajectory may contain shortcut actions that accidentally lead to the correct outcome without sufficient support from environmental information [62, 131]. Learning from such actions may therefore lead to guessing, confusion, and misuse at test time, ultimately causing adaptation to fail. To address this issue, we introduce EVIA, which identifies and corrects shortcut actions in trajectories to support more reliable action selection at test time.

By jointly studying how task knowledge can be represented, generated, retrieved, evaluated, verified, and adapted, this dissertation develops a unified perspective on reliable decision-making in interactive agents. Specifically, it includes methods that construct and select context-aware natural-language knowledge, evaluate agent robustness to imperfect external information, actively acquire missing knowledge through exploration, enable effective adaptation through functional decomposition and selective external tool use, and identify and correct actions that lack sufficient evidential support. Collectively, these contributions advance the development of interactive agents that learn more sample-efficiently, use acquired and external knowledge more robustly, and adapt more reliably to unfamiliar, uncertain, and continuously evolving environments.

1.1 Summary of Research Contributions

The core chapters of this dissertation are largely adapted from peer-reviewed papers published during my doctoral studies, as well as one manuscript currently in preparation. These works were conducted in collaboration with my co-authors. The main research contributions of this dissertation and their corresponding publication sources are summarized below.

1.1.1 Automated Generation and Selection of Context-Aware Guidelines for Large Language Model Agents (Chapter 2)

Recent advances in large language models (LLMs) have empowered AI agents capable of performing various sequential decision-making tasks. However, effectively guiding LLMs to perform well in unfamiliar domains such as web navigation, where they lack sufficient knowledge, has proven difficult under the demonstration-based in-context learning paradigm. In this chapter, we introduce a novel framework, called AutoGuide, which addresses this lim-

itation by automatically generating context-aware guidelines from offline experiences. Importantly, each context-aware guideline is expressed in concise natural language and follows a conditional structure, clearly describing the context in which it is applicable. As a result, our guidelines facilitate the provision of relevant knowledge for the agent's current decision-making process, overcoming the limitations of the conventional demonstration-based learning paradigm. Our evaluation demonstrates that AutoGuide significantly outperforms competitive baselines in complex benchmark domains, including real-world web navigation. This chapter is adapted from our prior work [32]. The source code for this chapter are publicly available at: https://github.com/lyney06/AutoGuide.

1.1.2 Beyond Blind Following: Evaluating Robustness of LLM Agents under Imperfect Guidance (Chapter 3)

Large language models (LLMs) have shown strong capabilities as task-solving agents across interactive domains. However, in complex environments, these agents may need to rely on auxiliary guidance to reduce the search space or compensate for limited domain-specific knowledge. Such guidance includes human-provided manuals and demonstrations, retrieved examples from memory or external tools, high-level heuristics, and agent-acquired knowledge from prior interactions. However, this guidance may be imperfect. For example, due to changes in the environment, ambiguous or simplified language, or retrieval errors from external sources, guidance can be incomplete, outdated, or contextually mismatched, potentially causing errors or failures during task execution. Building on our prior work [35], in this chapter, we introduce MIRAGE, a benchmark for Measuring Robustness of LLM Agents under Imperfect GuidancE. MIRAGE includes procedurally generated environments in navigation, cooking, and gaming, where both the environment and the auxiliary guidance vary in fidelity and relevance. We further extend MIRAGE to realistic web tasks via WebArena, using noisy or underspecified instructions extracted from demonstrations. Our findings reveal critical failure modes in current LLM agents and motivate future work on improving their robustness under imperfect guidance. The source code and datasets for this chapter are publicly available at: https://github.com/lyney06/MIRAGE.

1.1.3 Go Beyond Imagination: Maximizing Episodic Reachability with World Models (Chapter 4)

Efficient exploration is a challenging topic in reinforcement learning, especially for sparse-reward tasks. To deal with reward sparsity, researchers commonly apply intrinsic rewards

to motivate agents to explore the state space efficiently. In this chapter, we introduce a new intrinsic reward design called GoBI, or Go Beyond Imagination, which combines the traditional lifelong novelty motivation with an episodic intrinsic reward designed to maximize stepwise reachability expansion. More specifically, we apply learned world models to generate predicted future states with random actions. States with more unique predictions that are not in episodic memory are assigned high intrinsic rewards. Our method greatly outperforms previous state-of-the-art methods on 12 of the most challenging MiniGrid navigation tasks and improves sample efficiency on locomotion tasks from the DeepMind Control Suite. Building on our prior work GoBI [33], this chapter draws further inspiration from its underlying exploration principles to investigate why LLM-based agents that primarily prioritize task completion often perform poorly in shifted test-time environments and to develop test-time exploration strategies for improving agent performance. The source code for this chapter are publicly available at: https://github.com/lyney06/GoBI.

1.1.4 Hierarchical Procedural Meta-Reasoning for Generalizable Multimodal Agents (Chapter 5)

While multimodal agents can achieve strong performance through fine-tuning, their ability to generalize remains limited in complex real-world tasks such as mobile navigation, where diverse applications, frequent system changes, and customized workflows are common. We argue that a fundamental bottleneck lies in whether an agent possesses sufficient task-specific procedural knowledge to accomplish a given goal. In practice, due to the limited or outdated knowledge of the agent, the procedural steps it generates can be hallucinated and misaligned with the environment during execution. However, better procedural knowledge can be provided by the general capabilities of large language models or obtained from additional external resources, such as web search, when necessary. Based on this view, we propose the Procedure-Aware Multimodal Agent with Meta Reasoning, a framework that explicitly represents task knowledge as natural-language procedures and trains a procedure-aware grounded agent to condition its actions on this knowledge. By learning to leverage procedural knowledge from different sources, our approach enables robust and reliable generalization with reduced procedural hallucination across tasks, applications, interface versions, and multi-app workflows, achieving substantial improvements on challenging Android benchmarks. This chapter is adapted from our prior work [34].

1.1.5 Evidence-Grounded Verification for Interactive LLM Agents (Chapter 6)

Task completion is a commonly used evaluation metric for LLM-based agents. However, even within a trajectory deemed successful, not every action is necessarily evidence-grounded or self-contained. Such cases may arise when an LLM hallucinates or fails to verify important information yet still succeeds by chance, or when human demonstrators with strong environment-specific priors rely on internal knowledge and omit steps that would otherwise be necessary. In this chapter, we refer to such behaviors as epistemic shortcuts, which can cause direct failures at test time or create confusion during agent learning when they appear in training data. To address this issue, we propose Evidence-Grounded Verification for Interactive LLM Agents (EVIA), featuring an epistemic monitor that checks action-level evidence sufficiency by decomposing a proposed action into factual prerequisites and verifying whether those prerequisites are grounded in the observed trajectory. It supports both test-time control, by redirecting unsupported actions to alternative actions, and training-time rectification, by converting pseudo-successful trajectories into evidence-grounded demonstrations. Our evaluation demonstrates that EVIA outperforms baselines across multiple LLM-based agent domains. This chapter is based on a manuscript in preparation co-authored with Huijie Tang, Run Peng, and Honglak Lee.

CHAPTER 2

Automated Generation and Selection of Context-Aware Guidelines for Large Language Model Agents

2.1 Introduction

Recent advances in large language models (LLMs) have empowered AI agents to address various sequential decision-making tasks and applications [152, 161]. The foundation of these successes involves the planning and reasoning capabilities of pre-trained LLMs, enabling agents to execute effective policies [11, 159]. The predominant approach to leveraging these (typically closed-source) models for sequential decision-making tasks is to provide demonstrations in the form of in-context examples. However, direct application of this learning paradigm can be limited, especially in target domains where the LLM has insufficient prior knowledge such as in web navigation, where LLM agents generally achieve low success rates due to diverse and dynamic content [66, 22, 42, 188]. Providing all available experiences as demonstrations to an agent can further be unsuccessful due to context length limitations, prompt sensitivity, and difficulty with complex reasoning [89, 23, 95, 59].

On the other hand, LLMs excel in interpreting concise instructions provided as natural language, an ability that is also reinforced in the instruction-tuning phase of LLMs. Inspired by this, we explore data-driven strategies that leverage offline experiences to extract actionable knowledge to help guide LLM agents. As offline experiences implicitly convey valuable knowledge about desirable and undesirable policies in domains, they promise to serve as a useful resource for improving an LLM agent's decision-making in situations where the pre-trained LLM lacks understanding. Despite this potential benefit, a critical challenge lies in effectively extracting the implicit information embedded in offline data.

To address the challenge of extracting knowledge from offline data, we propose a novel framework, called AutoGuide. Specifically, AutoGuide automatically derives a comprehensive set of context-aware guidelines from offline experiences. Our method applies these context-conditional guidelines to enhance the performance of an LLM agent by retrieving

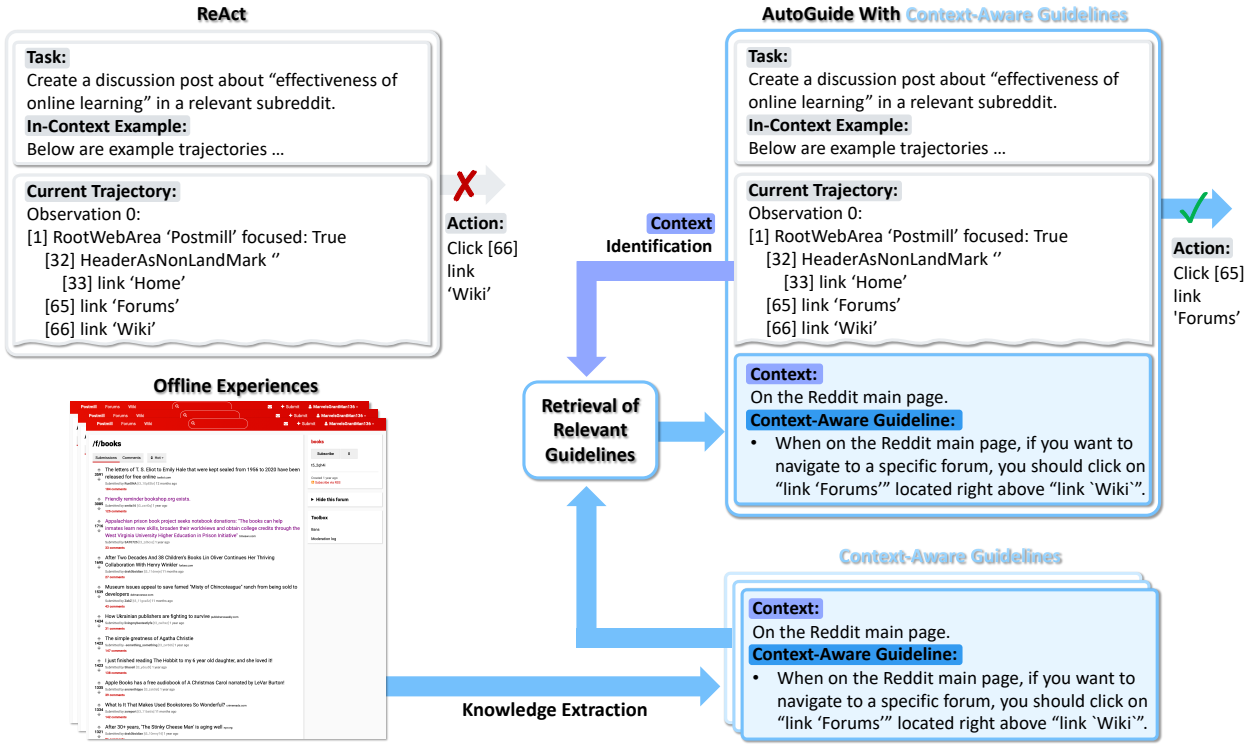


Figure 2.1: AUTOGUIDE aims to extract the implicit knowledge embedded in offline experiences and help the decision-making process of an LLM agent. Specifically, our method generates a comprehensive set of context-aware guidelines from offline data and explicitly identifies when each guideline is applicable by generating its corresponding context. Our context-aware guidelines enable providing pertinent guidelines at test time by identifying the context of the current trajectory, leading to correct decision-making compared to baselines without context-aware guidelines.

guidelines relevant to the agent's current state and incorporating them into the prompt during testing (see Figure 2.1). Notably, we generate context-aware guidelines in concise natural language statements, effectively compressing knowledge in offline data. Moreover, context-aware guidelines clearly describe the contexts where they are applicable, so AUTOGUIDE enables an LLM agent to select pertinent guidelines for its current decision-making process. As a result, AUTOGUIDE achieves the highest success rates compared to competitive baselines in complex sequential decision-making benchmark environments.

Our contributions. In summary, we present the following main contributions in this chapter:

- **Principled method based on context-aware guidelines (Section 2.3):** We develop two modules to automatically generate context-aware guidelines from offline experiences: the context identification module for identifying the context of a given trajectory, and the guideline extraction module for extracting a desired guideline corresponding to that context. The outcome is a set of domain knowledge in concise natural language that enhances decision-

making by providing pertinent information.

- **Comprehensive evaluation of AutoGuide (Section 2.4.2):** We show AutoGuide's capability in extracting helpful context-aware guidelines in various interactive benchmark domains, including navigating real-world web domains (e.g., GitHub). Our results highlight the effectiveness of AutoGuide, which significantly outperforms baselines without context-aware guidelines.

- **Analyses with important perspectives (Section 2.4.3):** We study various aspects of AutoGuide, such as the significance of determining the applicability of each guideline based on generated contexts. We also investigate the generalization ability of context-aware guidelines and demonstrate that our guidelines enhance the performance across out-of-domain tasks.

2.2 Related Work

LLM-based agents. Language models have recently been shown to possess strong priors for sequential decision-making tasks, which has given rise to LLM-powered agents [152, 161, 185, 174]. Agents need to possess various skills to be effective in practice including planning [11, 55, 88], reasoning [159, 37], tool manipulation [117, 109, 106, 127], code generation [141, 88], among others. In this chapter, we focus on building effective agents for web [170, 188] and embodied [135] environments.

Self-reflection from past experiences. An important capability for agents to succeed is the ability to learn from past experiences and update their behavior based on feedback. Self-feedback [92, 65, 132] has emerged as an effective technique where a model inspects its own incorrect predictions, reflects on them to identify what went wrong and attempts to improve its prediction. While self-feedback provides intra-task (i.e., per-episode) knowledge within a task based on immediate feedback, our approach offers an orthogonal and complementary aspect of inter-task knowledge (over multiple tasks) by considering multiple train tasks in offline data. AutoGuide enhances learning efficiency and credit assignment by utilizing detailed feedback from multiple tasks. However, these self-feedback approaches are complementary to AutoGuide and can be used in conjunction with our approach, as shown in our experiments (see Section 2.4.2).

Leveraging natural language guidance. Natural language can be a rich source of information for agents to learn to act efficiently. Prior work has explored the notion of learning from human-written text manuals, which describe details about the environment [9, 48, 186]. Recent work has explored automatically generating such guidance in the form

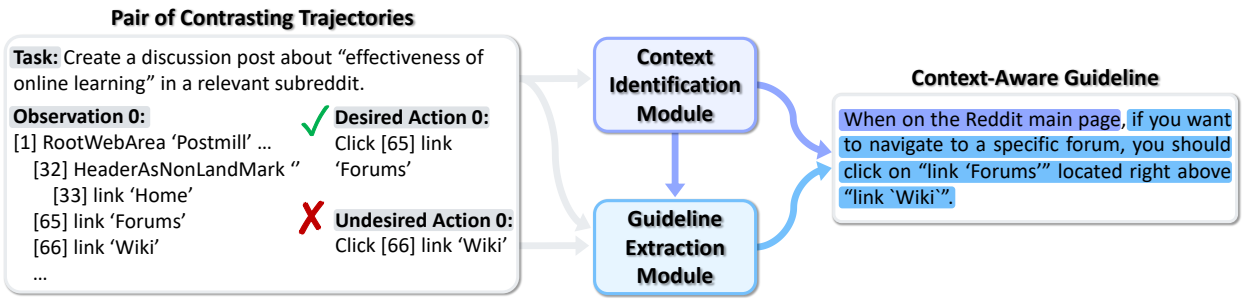


Figure 2.2: Context-aware guideline generation process based on a pair of contrastive trajectories τ_+^i and τ_-^i. In this example, the two trajectories start deviating from each other at $t=0$. The context identification module generates a description of the context at $t=0$ given $\tau_{:0}^i$, and the guideline extraction module generates the corresponding guideline for that context.

of chain-of-thought reasoning [159, 170], which emulates a thought process or rationale for agent's predictions. In contrast to approaches which generate such guidance dynamically on the fly by imitating example guidance demonstrations provided by a human, our approach carefully compares trajectories in offline data to generate appropriate guidance and uses these guidelines for predicting better actions. ExpeL [184] proposed a related approach to derive guidelines. In contrast to ExpeL, where all guidelines are provided to an agent as a prompt, our guideline selection process is contextual, where guidelines relevant to the agent's current state are retrieved and used for prediction. We show that this substantially improves over ExpeL's non-contextual guideline-based approach.

2.3 AutoGuide: Principled Method Based on Context-Aware Guidelines

Our work is motivated by the increasing availability of offline experiences that agents or humans naturally accumulate through their interactions with the environment. AutoGuide aims to leverage this offline data to improve the decision-making of an LLM agent by generating helpful context-aware guidelines. This section details how AutoGuide automatically constructs these guidelines and applies them to guide action generation at test time.

2.3.1 Problem Statement

Formally, AutoGuide is given offline data $\mathcal{D}_{\text{train}} = (\tau^1, ..., \tau^N)$ that consists of N trajectories from training tasks. Each trajectory $\tau = (x_0, a_0, r_0, ..., r_T)$ is a sequence of observations, actions, and rewards following the partially observable Markov decision process [144]. The return of a trajectory is defined as the sum of rewards obtained throughout the trajectory:

$R(\tau) = \sum_{t=0}^{T} r_t$. The objective of AUTOGUIDE is to distill knowledge from offline experiences into a useful natural language format, such that the extracted information helps to maximize the expected return $\mathbb{E}_\tau[R(\tau)]$ during test time.

2.3.2 Extraction of Context-Aware Guidelines

AUTOGUIDE generates a set of context-aware guidelines by utilizing pairs of contrastive trajectories from offline data. Each context-aware guideline is expressed in concise natural language and follows a conditional structure, clearly describing the context in which the guideline is applicable. Intuitively, contrasting a pair of trajectories with different returns provides important information about when and which actions are effective or ineffective in maximizing expected returns. Building on this insight, we develop two modules for automatically extracting context-aware guidelines (see Figure 2.2):

Context identification module. This module is responsible for abstracting the given partial trajectory into its context, a concise natural language description of the agent's state. More specifically, for a timestep t and the corresponding trajectory $\tau^i_{:t} := (x_0, a_0, ..., x_t)$, we prompt LLMs to clearly describe the agent's status:

$$\text{CONTEXT} \leftarrow \mathcal{M}_{\text{context}}(\tau^i_{:t}), \tag{2.1}$$

Our prompt templates for the context identification module are shown in Section A.3.1.

Guideline extraction module. This module aims to generate a desired guideline corresponding to a specific context. Let τ^i_+ and τ^i_- represent a contrasting pair of trajectories for the same task i in offline data $\mathcal{D}_{\text{train}}$, where $R(\tau^i_+) > R(\tau^i_-)$. We want to contrast the pair of trajectories to find desired behaviors at an important timestep. To do this, we compare these two trajectories to find the deviation timestep t at which they begin to diverge due to different actions. Then we apply the context identification module to summarize the context for the shared part of the trajectory $\tau^i_{:t}$. Eventually, we extract a useful natural language guideline by examining the paired contrastive trajectories τ^i_+ and τ^i_- with respect to the context:

$$\text{GUIDELINE} \leftarrow \mathcal{M}_{\text{guideline}}(\tau^i_+, \tau^i_-, \text{CONTEXT}), \tag{2.2}$$

where we refer to Section A.3.2 for our prompt template. As an example, the paired trajectories in Figure 2.2 deviate from timestep $t = 0$, for which the context is summarized as *On the Reddit main page*. This module then generates the following context-aware guideline: *When on the Reddit main page, if you want to navigate to a specific forum, you should click*

Algorithm 1 Extracting context-aware guidelines from offline data

Input: Offline data $\mathcal{D}_{\text{train}}$, context identification module $\mathcal{M}_{\text{context}}$, guideline extraction module $\mathcal{M}_{\text{guideline}}$
Initialize context-aware guideline dictionary $\mathcal{G}$
for Each pair $\tau_+^i, \tau_-^i \in \mathcal{D}_{\text{train}}$ **do**
  # Identify the context from a trajectory
  Find the deviating timestep t from τ_+^i and τ_-^i
  CONTEXT $\leftarrow \mathcal{M}_{\text{context}}(\tau_{:t}^i)$
  # Check if the current context matches any existing contexts
  if CONTEXT $\notin \mathcal{G}$ **then**
    $\mathcal{G}[\text{CONTEXT}] = \{\}$
  end if
  # Generate the context-aware guideline
  GUIDELINE$\leftarrow \mathcal{M}_{\text{guideline}}(\tau_+^i, \tau_-^i, \text{CONTEXT})$
  $\mathcal{G}[\text{CONTEXT}] \leftarrow \mathcal{G}[\text{CONTEXT}] \cup \{\text{GUIDELINE}\}$
end for
Return Context-aware guideline dictionary $\mathcal{G}$

Algorithm 2 Applying context-aware guidelines at test time

Input: Context-aware guideline dictionary $\mathcal{G}$, context identification module $\mathcal{M}_{\text{context}}$, guideline selection module $\mathcal{M}_{\text{select}}$, LLM agent policy π
Initialize test trajectory $\tau = \{x_0\}$
for Each timestep t **do**
  # Identify the current context from a trajectory
  CONTEXT $\leftarrow \mathcal{M}_{\text{context}}(\tau)$
  # If the current context matches any existing ones, perform top-k guideline selection
  if CONTEXT $\in \mathcal{G}$ **then**
    GUIDELINES $\leftarrow \mathcal{M}_{\text{select}}(\text{CONTEXT}, \tau; \mathcal{G}, k)$
  else
    GUIDELINES $\leftarrow \varnothing$
  end if
  # Action selection based on guidelines
  $a_t \sim \pi(\tau, \text{CONTEXT}, \text{GUIDELINES})$
  Execute action a_t and observe x_{t+1}
  Update trajectory $\tau \leftarrow \tau \cup \{\text{CONTEXT}, a_t, x_{t+1}\}$
end for

on "link 'Forums'" located right above "link 'Wiki'".

Construction of context-aware guidelines. We collect context-aware guidelines $\mathcal{G}$ by iterating through available pairs in the paired offline data and organize the guidelines in a dictionary format, using the context as the key and the corresponding guidelines as the value (see Algorithm 1). In particular, we observe that the context identification module occasionally produces contexts that describe the same situation but are expressed slightly differently. To minimize redundancy, we employ an LLM to determine if the current context corresponds to any previously identified context. If a match is found, we reuse the existing context; otherwise, we introduce a new context into our dictionary $\mathcal{G}$. The specific prompt template for this context-matching procedure is in Section A.3.3.

2.3.3 Applying Context-Aware Guidelines at Test Time

After extracting a set of context-aware guidelines $\mathcal{G}$ from offline experiences, our method employs these guidelines to enhance the decision-making of an LLM agent during testing. At each timestep, AUTOGUIDE identifies the CONTEXT of the current test trajectory τ up to

timestep t (which represents the agent's interactions up to the current time-step) using our context identification module $\mathcal{M}_{\text{context}}$. Our guideline selection module $\mathcal{M}_{\text{select}}$ then selects relevant guidelines for CONTEXT from $\mathcal{G}$. More specifically, the module applies CONTEXT as the key to fetch a set of possible guidelines $\mathcal{G}[\text{CONTEXT}]$. If there are more than k guidelines in $\mathcal{G}[\text{CONTEXT}]$, $\mathcal{M}_{\text{select}}$ prompts an LLM to choose top-k guidelines for the specific τ:

$$\text{RELEVANT GUIDELINES} \leftarrow \mathcal{M}_{\text{select}}(\text{CONTEXT}, \tau; \mathcal{G}, k), \tag{2.3}$$

where Section A.3.4 details the prompt template for this selection procedure. Subsequently, AUTOGUIDE incorporates both the context and relevant guidelines into the agent's action generation prompt. Therefore, the agent selects an action by considering the provided context and guidelines (see Figure 2.1 for an example). This process iterates until the end of the test trajectory (see Algorithm 2).

Key benefits of AutoGuide. First, the extraction of context-aware guidelines in AUTOGUIDE offers the inherent benefit of providing relevant guidelines for the context of interest. This capability is important since neglecting the specific context in which a guideline applies can confuse the agent's decision-making process. The second key benefit is the generation of concise natural language guidelines, which can be seamlessly incorporated into any prompt-based LLM agent. Lastly, AUTOGUIDE generates guidelines at the individual context level rather than at the trajectory level. Given that a single incorrect action can lead to a complete failure, it is essential to provide detailed assistance in each action selection process. With these advantages, we demonstrate in the next section that our approach significantly enhances the performance of LLM agents.

2.4 Evaluation

This section demonstrates the efficacy of AUTOGUIDE by conducting experiments on a diverse suite of sequential decision-making benchmark domains. We also perform important analyses about AUTOGUIDE, such as the ablation study of different AUTOGUIDE components, comparison to in-context learning, and generalization to out-of-domain tasks. We refer to Section A.2 for additional experimental details.

2.4.1 Evaluation Setup

We describe the benchmark domains, baseline models, and experimental setup used to evaluate AUTOGUIDE.

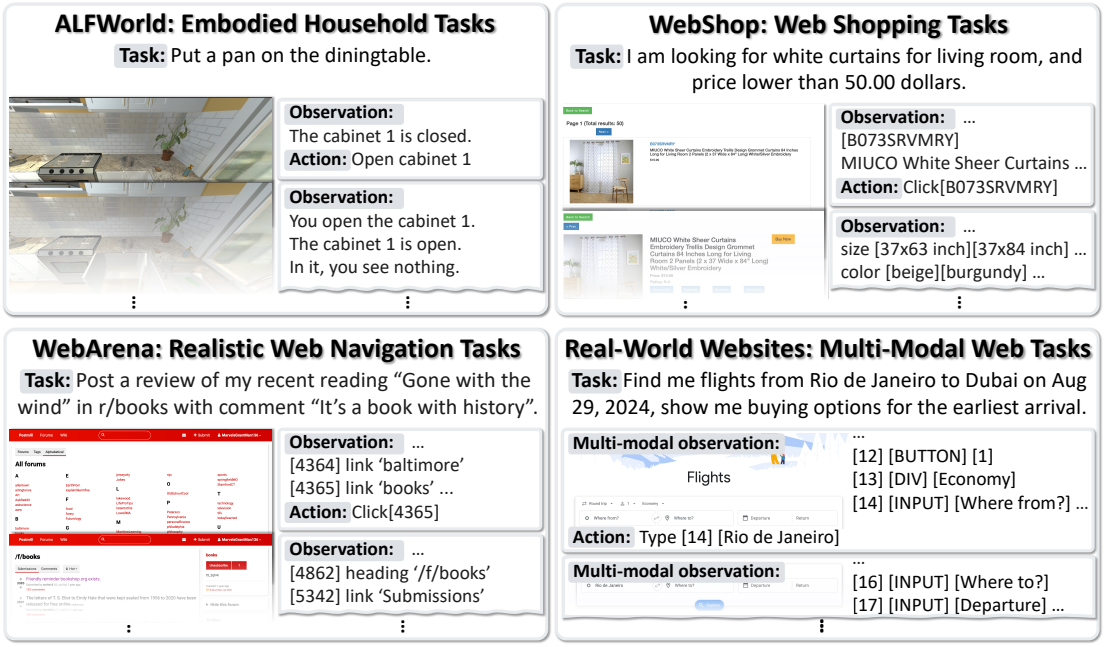


Figure 2.3: Sequential decision-making benchmark domains considered in our work: ALFWorld [135], WebShop [169], WebArena [188], and multimodal real-world websites. Graphic credit: [134, 169, 188, 1].

2.4.1.1 Sequential Decision-Making Benchmark Domains

We consider the following interactive sequential decision-making benchmarks to study various aspects of AutoGuide (see Figure 2.3):

- **ALFWorld [135]:** In this embodied benchmark, an LLM agent interacts with an environment to carry out household tasks, such as placing a pan on the dining table. Observations and actions are expressed in natural language statements, and the agent must navigate through the space and manipulate objects to successfully complete the tasks.

- **WebShop [169]:** This interactive web environment simulates the task of online shopping on an e-commerce website. The agent's goal is to understand a text instruction and buy a product that meets specified criteria. This involves querying the website's search engine, understanding the descriptions and details of each item, and selecting necessary options.

- **WebArena [188]:** This web-based benchmark introduces realistic environments by replicating the functionality and data found in popular web domains (e.g., GitLab, Reddit, Wikipedia). Compared to WebShop, WebArena presents more challenges and difficulties for an LLM agent due to its large observation and action space, along with tasks that involve longer planning horizons. We focus on the Reddit domain for the main WebArena experiments.

• **Real-world multimodal websites:** Finally, we consider evaluating AutoGuide on a variety of real-world website tasks. These span from a collaborative software development platform (e.g., GitHub) to a flight search engine (e.g., Google Flights) and an online education platform (e.g., Coursera). Please refer to Section A.2.4 for example tasks. In particular, in comparison to WebShop and WebArena, we design our tasks to be multimodal such that the agent must consider both visual (e.g., images) and textual information (e.g., HTML) to complete these tasks.

2.4.1.2 Baselines

We compare AutoGuide against the following baseline approaches to study the effect of context-aware guidelines (refer to Section A.2 for more details):

• **ReAct [170]:** This LLM-based planning method integrates reasoning and acting to address sequential decision-making tasks. However, it does not leverage offline experiences and thus suffers from the limited understanding of pre-trained LLMs in downstream domains.

• **ExpeL [184]**: This method also extracts natural language knowledge from offline data. However, it fails to consider the applicability of guidelines and does not generate context-aware guidelines. Instead, it provides all guidelines to an LLM agent without filtering out irrelevant ones based on the current context. ExpeL has two contributions, the guideline generation and in-context example selection module. Because the latter is orthogonal to our analysis and can be seamlessly combined with our method, we consider ExpeL with guidelines in our experiments.

• **Reflexion [132]:** This approach converts environmental feedback into text statements to assist an LLM agent (e.g., ReAct) in the next trial. The baseline generates valuable feedback about solving a specific test task. We demonstrate how context-aware guidelines derived by AutoGuide can be combined with the feedback.

2.4.1.3 Implementation

We collect offline experiences either by running ReAct and Reflexion, or incorporating human demonstrations. We use ReAct with GPT-3.5-turbo as our base LLM agent for WebShop and ALFWorld and GPT-4-turbo for WebArena. For each benchmark, we apply the same GPT model for action generation, context identification, and guideline selection. We extract context-aware guidelines from offline data with GPT-4-turbo and evaluate their effectiveness by applying them to the test set with non-overlapping tasks. We refer to Section A.2 for more details.

Algorithm	Offline data?	Context aware?	ALFWorld [135]	WebShop [169]		WebArena [188]
			Success Rate (SR)↑	Reward↑	SR↑	SR↑
ReAct [170]	✗	✗	54.5%	66.4	30%	8.0%
ExpeL [184]	✓	✗	59.0%	60.9	35%	21.8%
AutoGuide	✓	✓	**79.1%**	**73.4**	**46%**	**47.1%**
ReAct [170] + Reflexion [132]	✗	✗	67.2%	77.1	51%	N/A
ExpeL [184] + Reflexion [132]	✓	✗	71.6%	71.7	42%	N/A
AutoGuide + Reflexion [132]	✓	✓	**88.1%**	**81.4**	**57%**	N/A

Table 2.1: Test reward and success rate on ALFWorld, WebShop, and WebArena. The base agent model for ALFWorld and WebShop is GPT-3.5-turbo and for WebArena is GPT-4-turbo. Reflexion is done by GPT-4-turbo for at most 3 trials. In our experiments, due to token limit of GPT, we did not experiment with Reflexion on WebArena tasks.

2.4.2 Main Results

Q1. *How effective is AutoGuide compared to baselines without context-aware guidelines?*

To answer this question, we compare methods on ALFWorld, WebShop, and WebArena benchmarks. The performance on the test datasets is presented in Table 2.1. There are three notable observations:

1. **Effectiveness of context-aware guidelines.** Our approach surpasses baseline performance in both ALFWorld and WebShop, achieving the highest test rewards and success rates in Table 2.1. These results highlight the effectiveness of employing context-aware guidelines in language-based decision-making domains. To further examine the action selection differences among ReAct, ExpeL, and our method, we present their trajectories in Figure 2.4. We observed that ReAct makes common mistakes such as trying to take a soapbar that is not visible, or taking a soapbottle instead of a soapbar due to their similar names. Both ExpeL and AutoGuide improve on this by extracting guidelines from similar mistakes in the offline experience. However, ExpeL often erroneously applies incorrect guidelines due to the availability of all guidelines at each timestep. In Figure 2.4, ExpeL mistakenly attends to the second guideline "ensure to specify the item's number and location ...", leading to wrong reasoning and action. AutoGuide presents relevant guidelines at necessary moments, enabling accurate task completion by avoiding the mistakes seen in ExpeL and ReAct.

2. **Importance of providing pertinent knowledge.** ExpeL helps ReAct by extracting knowledge from offline experiences, but its impact is not as significant as AutoGuide. Recall that for ExpeL, the guidelines are neither generated for specific contexts at training time nor selected to only provide context-aware guidelines at test time. As a result, irrelevant guidelines can be introduced to an agent, potentially causing confusion for the agent. Con-

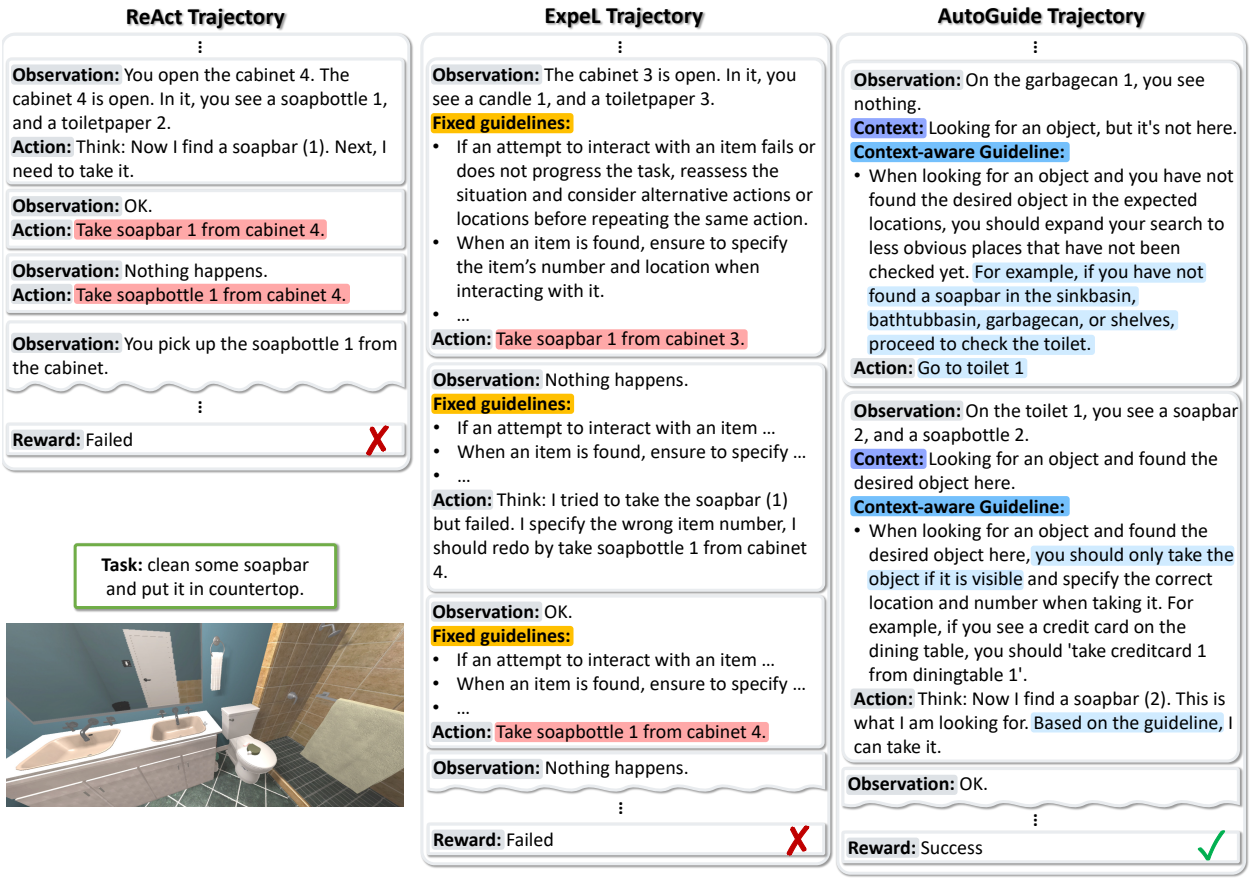


Figure 2.4: Trajectories of ReAct, ExpeL, and AutoGuide from the same test task. ReAct (Left) chose the wrong item, consequently failing the task in the end. ExpeL (middle) was confused by guidelines that were irrelevant to current context, leading to incorrect reasoning and actions. AutoGuide (right) selects relevant guidelines to the agent's context, enabling the agent to accomplish the task.

sequently, the result highlights the significance of providing relevant guidelines conditioned on contexts for LLM agents.

3. **Scalability to complex environments.** We conduct experiments on WebArena-Reddit, which features more diverse tasks on realistic and complex websites requiring longer action sequences. This domain has a larger observation space and a more complex action space (e.g., scrolling). Table 2.1 presents the results, where AutoGuide achieves the highest success rate with a significant margin when compared to ReAct and ExpeL. We observe that ReAct scores low task success rate (8.0%) in WebArena due to the complex observation and action spaces and longer task horizon. In ExpeL, the issue of presenting all guidelines to an agent is exacerbated in the WebArena compared to simpler environments like ALFWorld and WebShop. WebArena's wide variety of tasks across different domains requires a larger number of guidelines to cover the knowledge needed for all tasks and domains. This results

Algorithm	GitHub	Flights	Coursera
	SR↑	SR↑	SR↑
SoM [167]	2/30	5/20	1/20
AutoGuide	**19/30**	**9/20**	**14/20**

Table 2.2: Test results of AutoGuide on 3 real-world web domains within multimodal settings. The base agent model runs with GPT-4V and applying context-aware guidelines significantly improves the performance.

Algorithm	WebShop	
	Reward↑	SR↑
ReAct (1-shot)	66.4	30%
ReAct (2-shot)	66.0	35%
ReAct (4-shot)	70.2	37%
ReAct (6-shot)	71.0	38%
AutoGuide	**73.4**	**46%**

Table 2.3: Analysis of AutoGuide against ReAct with varying numbers of in-context examples.

Top-k	WebShop
	SR↑
$k=0$	30%
$k=1$	42%
$k=2$	46%
$k=3$	47%
$k=5$	43%

Table 2.4: Ablation study of AutoGuide using various top-k values.

in either an overload of irrelevant guidelines that could mislead the agent or a lack of crucial information when the number of guidelines is limited, as suggested in ExpeL [184]. In contrast, AutoGuide achieves a more significant performance enhancement (47.1%) compared to ExpeL (21.8%) by efficiently providing pertinent guidelines and minimizing the burden on context capacity. We refer to Figure A.10 for a list of example contexts and guidelines.

Q2. *How does AutoGuide perform when combined with test-time self-feedback approaches?*

Our context-aware guidelines effectively provide *inter-task* knowledge by considering multiple tasks in offline data. Meanwhile, self-feedback methods (e.g., Reflexion) offer *intra-task* knowledge based on environmental feedback during test time. In this question, we explore the effectiveness of integrating both inter-task and intra-task information. The results presented in Table 2.1 demonstrate that the combination of AutoGuide with Reflexion achieves the highest performance in the WebShop and ALFWorld benchmarks. Hence, we find that our context-aware guidelines positively complement the intra-task knowledge of Reflexion. Another observation from Table 2.1 is that, while ExpeL + Reflexion outperforms ExpeL alone, this combination is not as effective as other approaches. This limitation may stem from ExpeL introducing irrelevant knowledge, potentially leading to conflicts with Reflexion's feedback and having an adverse impact on the decision-making process.

Q3. *Can AutoGuide generate context-aware guidelines for multimodal inputs?*

Going beyond text-only inputs is an essential step toward building capable agents for solving real-world environments and tasks. We test AutoGuide in a complex multimodal setting, where each observation includes image and text information. Specifically, we introduce a set of real-world website navigation tasks in 3 domains: GitHub, Google Flights, and Coursera. For these multimodal tasks, we employ the Set-of-Mark (SoM) agent [167, 66] as our base method. The SoM prompting improves the visual grounding capabilities of large

multimodal models such as GPT-4V by adding visually distinguishable marks to image inputs [167]. We apply AutoGuide with GPT-4V to generate natural language context-aware guidelines from collected trajectories with both image and text observations. Table 2.2 shows the effectiveness of AutoGuide, demonstrating its generalization ability to complex real-world multimodal settings. We refer to Figure A.11 for example context-aware guidelines.

2.4.3 Analyses of AutoGuide

Q4. *How does AutoGuide compare to ReAct with varying numbers of in-context examples?*

Table 2.3 shows that, while increasing the number of in-context examples for ReAct gradually improves performance, there is a plateau at a certain number of shots. Additionally, ReAct with more than 6 shots often exceeds the token limit of GPT-3.5-turbo. These results indicate that directly inputting raw trajectories into ReAct for in-context learning is not an effective way to fully leverage offline data. In contrast, AutoGuide extracts knowledge from entire training trajectories by summarizing them into concise context-aware guidelines, making them easy to integrate with prompt-based agents.

Q5. *How does altering the number of top-k guidelines impact the performance of AutoGuide?*

We conducted an ablation study on WebShop using various values of k in Table 2.4. We find that employing context-aware guidelines consistently outperforms the no-guideline baseline ($k=0$; ReAct). The $k=3$ yields the best performance. The largest k value of 5 can lead an LLM agent to overthink, potentially resulting in a slight decrease in performance. Conversely, a smaller k, like $k=1$, may cause LLM to overlook additional helpful guidelines, leading to slightly worse performance.

Q6. *How do AutoGuide's context-aware guidelines generalize to out-of-domain environments?*

We conduct an experiment to further demonstrate AutoGuide's out-of-domain capability across different domains but relevant tasks. We extract context-aware guidelines from WebShop and apply them to WebArena-Shopping, which is a distinct domain with variations in observation/action spaces, task intentions, and episodic horizons. For this domain adaptation case, we additionally incorporate a grounding module to align the context-aware guidelines from WebShop to WebArena's observations based on GPT-4-Turbo. As shown in Table 2.5, the transferred guidelines bring a notable improvement in success rates compared to the ReAct baseline in WebArena Shopping.

Q7. *How does each component of AutoGuide contribute to the final results?*

We evaluate the impact of different components within AutoGuide on its performance in

Algorithm	WebArena–Shopping
	SR↑
ReAct	10.2%
AutoGuide	20.4%

Table 2.5: Out-of-distribution generalization of context-aware guidelines from WebShop on the 98 WebArena–Shopping tasks that have a product in the intent template.

Algorithm	CI	GES	WebShop SR↑
ReAct	✗	✗	30%
ReAct + CI	✓	✗	36%
ReAct + GES	✗	✓	37%
AutoGuide	✓	✓	**46%**

Table 2.6: Ablation study of AutoGuide, analyzing each module's contribution in WebShop. CI denotes our context identification module, and GES denotes the guideline extraction and selection modules.

WebShop, as detailed in Table 2.6. We examine two variants: ReAct+CI and ReAct+GES. The ReAct+CI, which incorporates contexts into observations without guidelines, shows improvement over ReAct. This suggests that contexts enhance decision-making by verifying the current state before action selection. ReAct+GES, which generates guidelines from trajectories without contexts and employs GPT-3.5-turbo for guideline selection, also enhances performance but is less effective than the full AutoGuide. This indicates that choosing relevant guidelines based on the trajectory alone is more challenging than using contexts. Therefore, integrating both context summaries and guidelines is crucial for maximizing the benefits of AutoGuide.

2.5 Conclusion

We present AutoGuide, an effective framework for exploiting important domain knowledge from offline experiences for improving decision-making with pre-trained LLMs. We proposed to generate context-aware guidelines that can be incorporated into prompts for LLM agents. As AutoGuide extracts the guidelines by contrasting trajectories in offline data, the resulting context-aware guidelines carry critical information for preventing failures in the domains. For inference, it provides the guidelines pertinent to each of the different contexts that LLM agents encounter, which can make pre-trained LLMs strong decision-making agents in the downstream domains. We showed that AutoGuide outperforms strong baselines by a large margin and achieves outstanding performance in decision-making benchmarks.

CHAPTER 3

Beyond Blind Following: Evaluating Robustness of LLM Agents under Imperfect Guidance

3.1 Introduction

Recent advances in large language models (LLMs) have made them increasingly reliable at solving complex decision-making tasks in diverse interactive environments [161, 170, 85], including web navigation [178], embodied control [79], and open-ended games [104, 148]. To further enhance the effectiveness of LLM agents, many frameworks incorporate auxiliary guidance to provide situational context, narrow the search space, or supplement domain-specific knowledge. Such guidance may include retrieved manuals or demonstrations via retrieval-augmented generation (RAG) [60], reusable workflows [158], natural language cues and strategies acquired through prior interactions and memory [184, 32], or clarifications and hints from interactive dialogue with users [156, 98].

Unfortunately, this guidance is often imperfect, introducing critical challenges for agent robustness. Imperfections typically stem from two sources. First, guidance may be intrinsically flawed. For example, it may omit key steps or include vague descriptions. Such imperfections are common in real-world scenarios: human-written guidance may contain typos, omissions, or misremembered details, and LLM-generated guidance may include hallucinations, over-simplifications, or subtle misstatements of relevant information [54]. Second, guidance may be misaligned with the environment, especially in dynamic, noisy settings [105]. A plan that was once valid may no longer apply due to changes such as system updates, object movements, or seasonal variations, resulting in undesired actions.

Given the prevalence of imperfect guidance, a central research question emerges: *Can LLM agents detect and adapt to imperfect task-related guidance effectively?* To facilitate systematic study, we introduce **MIRAGE** (Measuring Robustness of LLM Agents under Imperfect GuidancE), a two-part benchmark suite comprising: (1) MIRAGE-World, a collection of procedurally generated environments spanning navigation, cooking, and gaming

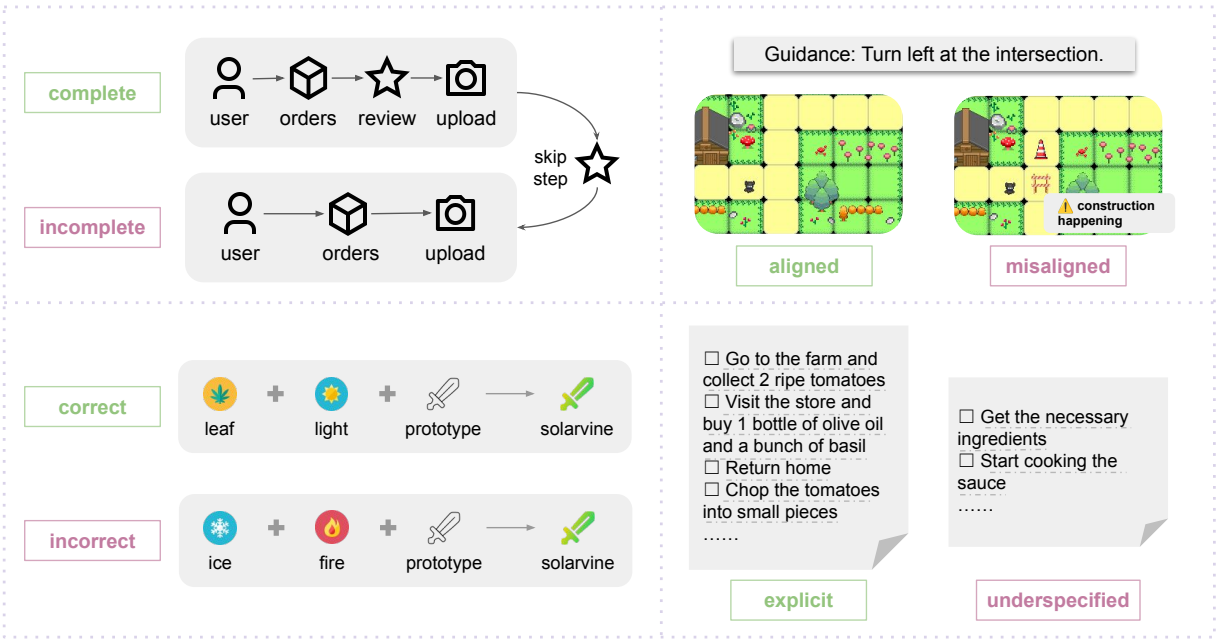


Figure 3.1: Examples of four types of guidance imperfection. (1) Incomplete: a critical step (e.g., uploading a photo to review an order) is omitted; (2) Incorrect: a weapon crafting recipe includes wrong components; (3) Misaligned: originally correct navigation guidance ("turn left") becomes invalid because the road is under construction; (4) Underspecified: cooking instructions are vague and high-level, lacking actionable detail.

tasks; and (2) MIRAGE-Web, a set of 450 realistic web-based tasks derived from the WebArena platform [188] across four domains (shopping, gitlab, reddit, and shopping admin). For both settings, we introduce controlled perturbations to the guidance and the environment to systematically evaluate agent robustness. We categorize guidance imperfections into four types based on how they are constructed: incomplete (critical steps are omitted), incorrect (false details are introduced), misaligned (environment changes invalidate previously correct guidance), and underspecified (vague or abstract phrasing obscures execution details). These imperfection types, illustrated in Figure 3.1, reflect failure patterns commonly encountered in real-world scenarios, and enable structured analysis of how well agents generalize and recover under imperfect guidance.

All tasks in MIRAGE are carefully constructed to be well-defined and solvable. Each task includes a clear and stable goal that does not change and is communicated unambiguously to the agent. What we vary is the auxiliary guidance provided to support task completion, not the task specification itself. By design, every task—including those with environment changes—remains solvable. For example, in navigation scenarios where a road is blocked due to construction, an alternative path is always available. Agents can succeed if they are able to reason about inconsistencies or adapt based on feedback from the environment.

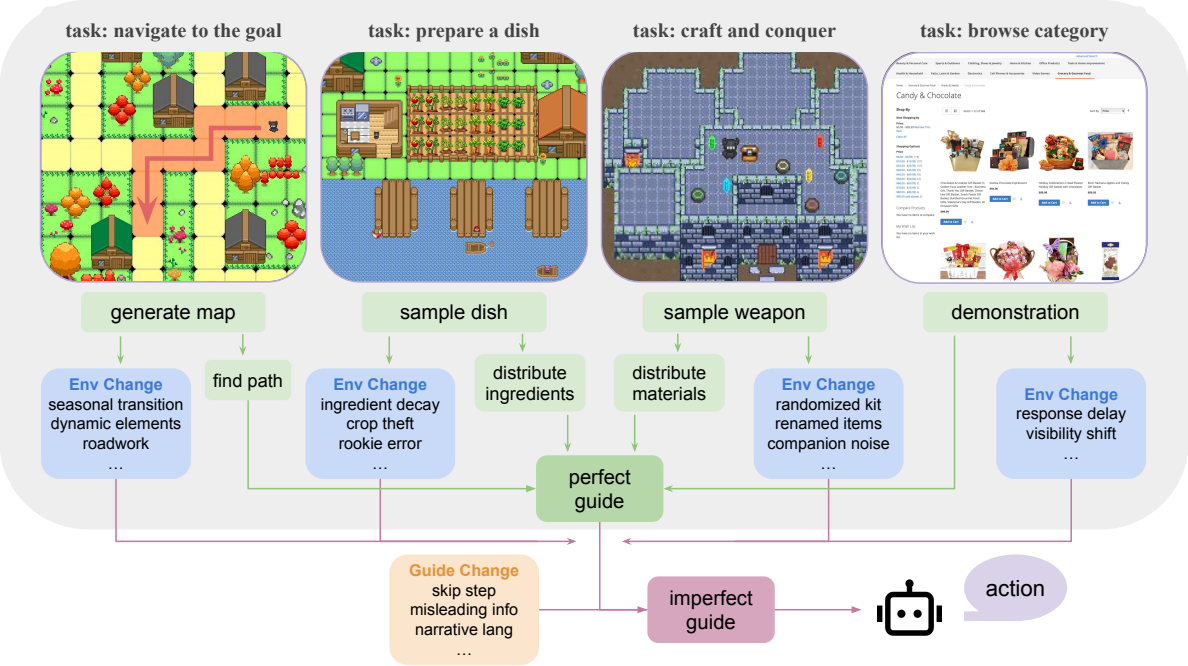


Figure 3.2: Overview of MIRAGE. For navigation, cooking, and gaming, we first generate the environment and task instance (e.g., map layout, recipe goal, or target boss). From this, we derive a perfect solution path, which is then optionally perturbed to produce an imperfect guide. For web tasks, we extract guidance directly from human demonstrations that we collected. We introduce two types of perturbations: Env Change (blue), which alters the environment itself, and Guide Change (orange), which corrupts the instruction.

In summary, MIRAGE provides a unified testbed for guidance robustness in LLM agents. Specifically, we summarize our contributions as follows:

- We introduce MIRAGE, a benchmark suite for evaluating LLM agent robustness under imperfect guidance. It includes over 450 real-world web navigation tasks derived from WebArena, and procedurally generated tasks across navigation, cooking, and gaming.
- The benchmark systematically injects four types of guidance imperfections—incomplete, incorrect, misaligned, and underspecified—via programmatic generation, LLM-based rewriting, and human verification, enabling controlled and compositional benchmarking.
- We conduct thorough evaluations using widely-adopted LLMs on MIRAGE. Results show that current LLM agents often struggle to detect flaws or adapt their behavior accordingly, highlighting the importance of guidance robustness for LLM agents.

3.2 Related Work

Instruction Following in LLMs. As LLMs become better at following instructions [51, 52], guidance retrieval has emerged as a promising approach for improving agent performance [184, 15], though its success still highly depends on the quality of the guidance. In practice, guidance is often incomplete, ambiguous, or misleading [111, 124, 140]. This limitation is particularly well-documented in domains such as embodied navigation [80], robotics [111], and autonomous driving [124], wherein misguided instructions from non-expert users can cause critical safety issues if blindly followed. Recent LLM research has begun studying imperfect instructions, primarily focusing on ambiguous user queries [156, 116, 5], showing that LLMs often make incorrect assumptions to compensate for ambiguity, and on robustness against adversarial attacks [110, 71, 165], which is crucial for safe deployment. MIRAGE expands these lines of inquiry by systematically benchmarking LLMs across multiple imperfection categories and diverse domains.

Reflection and Error Recovery. Self-reflective agents [132, 31, 187] aim to improve performance by iteratively revising plans or prompting models to reason about their past failures. While methods such as Reflexion [132] incorporate explicit mechanisms for self-reflection, recent findings suggest that these mechanisms are often brittle and unreliable, particularly in environments with noisy or misleading feedback [132, 148, 61]. At the same time, most of these approaches implicitly assume that errors stem from the agent's internal reasoning, rather than from flaws in external guidance. In contrast, our setting introduces external imperfections, which place greater demands on the agent's ability to question, revise, or recover from faulty inputs. This makes reflection not only desirable but essential: agents must learn not just to correct themselves, but to recognize when the information they were given is untrustworthy.

Benchmarks for Evaluating Robustness. A growing set of benchmarks evaluate LLM agents across web navigation, embodied control, and open-ended domains [85, 154, 135, 169, 104, 188]. However, these environments do not specifically test LLM agents' robustness under imperfect guidance. MINT [157] explores agent improvement in response to "lazy" user responses, which are similar to our notion of underspecified guidance. We expand upon MINT by exploring additional types of imperfections, and we focus on guidance presented prior to any attempt rather than feedback presented after an attempt. DOROTHIE [91] studies recovery under unexpected events in autonomous driving, but focuses on dialogue rather than single-agent execution. In contrast, MIRAGE introduces controlled guidance

imperfections spanning *incomplete, incorrect, misaligned*, and *underspecified* and supports structured evaluation of agent recovery behaviors.

3.3 MIRAGE Benchmark Framework

3.3.1 Problem Setting

In MIRAGE, each interactive task is paired with a clearly defined goal, such as reaching a destination or preparing a specific dish. We additionally provide *guidance*, which refers to any form of auxiliary information intended to assist the agent's decision-making. This may include textual manuals, hints, or language skills designed to support task completion.

In each task, the agent is given a `goal` and accompanying `guide`. At each timestep t, the agent receives an observation o_t and produces an action $a_t = \pi_{\text{LLM}}(o_t, \texttt{goal}, \texttt{guide})$. Our benchmark focuses on test-time settings where the guidance may be noisy or misleading, and evaluates how well agents can adapt to such imperfections.

3.3.2 Benchmark Overview

MIRAGE consists of two parts: MIRAGE-World, which includes procedurally generated environments in navigation, cooking, and gaming, and MIRAGE-Web, which builds on realistic web navigation tasks from WebArena [188]. As shown in Figure 3.2, each domain supports correct guidance that enables successful task completion. From these, we systematically derive its imperfect variants by either perturbing the guidance itself or modifying the environment to break its alignment. This design enables controlled comparison across both guidance fidelity and format, allowing us to evaluate agent robustness, isolate the effects of imperfection, and examine the performance gap between effective and suboptimal guidance. Table 3.1 gives a domain-level overview. Additional environment details are provided in Appendix B.2.

3.3.2.1 Task domains

Navigation. We design procedurally generated 12×12 grid-based maps to simulate rural navigation environments in which agents must reach specific destinations. Each map features fixed structural elements, such as houses, trees, and roads, which differ in layout across episodes. Agents receive a 3×3 partially observable view. Some instructions are precise and observation-independent (e.g., "move north for three tiles"), while others are abstract and depend on observations (e.g., "walk forward until you see a maple tree on your right").

Info	VillageNav	Cooking	Roguelike	WebArena
Task Goal	reach a destination in a village grid	gather ingredients and prepare dishes	craft a weapon and defeat the boss	complete realistic web-based user tasks
# Tasks	procedurally generated	procedurally generated	procedurally generated	450 manually curated
Incomplete	—	lacks ingredient loc	skip level info	skip action step
Incorrect	extra detour wrong step count	tool limits	invalid recipe	UI change
Under specified	—	simplified recipe	strategy sketch	naturalized guidance step merging
Misaligned	blocked road seasonal change dynamic obs	mismatched recipe storage loss rookie failure	renamed items randomized kit	response delay visibility shift

Table 3.1: Overview of MIRAGE, including task statistics and example imperfect guidance.

Cooking. Agents are tasked with preparing dishes by collecting and processing ingredients using various tools and appliances. They must navigate between the kitchen, grocery store, fishing dock, and farm to gather resources, and manage time effectively during cooking by parallelizing cooking steps. Guidance is provided through textual recipes and ingredient locations that vary in completeness and correctness.

Roguelike Gaming. This domain features multi-level dungeons in which agents must explore the environment, collect items, craft weapons, and defeat enemies. The provided guidance may be detailed, level-specific plans or general strategic instructions such as "craft Fire Staff before reaching the boss."

Web Navigation. Our web tasks are built on top of WebArena [188], a realistic and open-ended environment for benchmarking language agents on realistic websites. Each task specifies a user goal (e.g., "post a message on Reddit"), which the agent must achieve through a sequence of natural language actions. We focus on four representative domains: Reddit, Shopping, CMS (content management), and GitLab.

3.3.3 Guidance Imperfection

MIRAGE provides a unified framework for modeling and evaluating imperfect guidance across diverse domains. As illustrated in Figure 3.2, each task instance begins with a clearly defined goal and an associated reference guidance, which is sufficient to complete the task under ideal conditions.

To systematically introduce imperfections, we apply two complementary mechanisms:

Guide Change and Env Change. Guide Change modifies the content of the guidance itself, for example, by omitting steps, introducing incorrect substitutions, or abstracting specific actions. Env Change, on the other hand, alters the environment in ways that break alignment with otherwise correct guidance, such as changes in UI layout or blocked paths. This framework enables precise and controllable perturbations to the fidelity of the guidance. Since each type of imperfection is derived through a specific transformation of the original guidance, we define imperfection categories based on how the flawed guidance is constructed. Specifically, we categorize imperfections into the following types:

- **Incomplete**: Deliberately removing key steps or information (e.g., skipping the step to open a product page before adding it to the cart).
- **Incorrect**: Altering factual or procedural elements in the guidance (e.g., providing an invalid crafting recipe that does not produce any valid weapon).
- **Underspecified**: Merging, abstracting, or informalizing details in the original guidance. This category includes vague phrasing, high-level goals, and compressed multi-step instructions, requiring the agent to infer missing semantics from context.
- **Misaligned**: Guidance that becomes invalid due to changes in the environment (e.g., renamed materials or blocked roads), resulting in a task mismatch.

While these categories are clearly defined from a designer's standpoint, they may blur together from the agent's perspective. For instance, a skipped step (incomplete) may be perceived as an incorrect guide or a misaligned reference. Our taxonomy therefore emphasizes the source of perturbation rather than the agent's internal attribution, supporting systematic diagnosis of robustness under distinct failure conditions.

Table 3.1 summarizes the distribution of guidance imperfections across domains. These imperfections give rise to diverse challenges: from reasoning over missing or abstract information (e.g., multi-step actions such as "search", which clicking and typing), to grounding vague or inconsistent references in observations (e.g., ambiguous spatial cues in navigation), to adapting plans in response to failed execution (e.g., invalid crafting recipes). Although these imperfections are *challenging*, they are never *insurmountable*. MIRAGE consistently provides contextual clues that enable the agent to infer the existence of imperfections and correct them by backtracking and re-planning.

3.3.4 Environment Implementation and Task Construction

All environments in MIRAGE-World are powered by a fully custom simulation engine built from scratch, using Pygame [115] and CUTERPG pack [112]. Unlike prior work with fixed

layouts, our engine supports dynamic environment generation, variable goal configurations, and programmatic injection of guidance imperfections. These imperfections are introduced through strategies such as omitting key steps, inserting incorrect actions, or generating abstract instructions conditioned on local observations. The engine provides both a Gym-style API [10] for agent interaction and a graphical interface for human use and debugging. For MIRAGE-Web, we extend the WebArena framework [188] and curate 450 task trajectories from subsets of WebArena that are solvable by humans. Imperfections are introduced via a combination of hardcoded edits (e.g., step omissions) and LLM-prompted rewrites, followed by human review to ensure semantic consistency.

All tasks in MIRAGE are solvable by construction. During task generation, we first ensure that a valid solution path exists under perfect guidance. For example, in the gaming domain, we generate the target boss, then select a compatible weapon, distribute the necessary crafting materials across levels, and assign a health threshold that matches the weapon's effectiveness. This guarantees that a feasible solution exists before any imperfections are introduced. For tasks involving environment-level changes (e.g., blocked paths or missing items), we apply dedicated solvability verifiers as part of the generation pipeline to ensure that the perturbed variant remains solvable. As an additional sanity check, we conduct a human study in which five participants attempt a representative subset of tasks. While human performance is not used as a definitive criterion, it provides further confirmation that the tasks are interpretable and feasible for agents with sufficient reasoning ability.

3.3.5 Extensibility and Difficulty Scaling

A key strength of MIRAGE-World is its extensibility. As LLMs improve rapidly, our tasks can be scaled to preserve challenge and diagnostic value. All environments are procedurally generated and do not rely on fixed layouts or solution paths, enabling adjustable complexity across task instances. For example, navigation difficulty can increase through larger maps, added obstacles, or multi-goal objectives. Cooking tasks may involve multi-dish requirements, tool constraints, or more complex recipe structures. Roguelike can scale via deeper dungeons, more intricate crafting, and more complex battle systems. These extensions are easy to implement and ensure that MIRAGE remains relevant as LLM capabilities grow.

3.4 Experiments

We evaluate a mix of LLM models and agent frameworks. In `MIRAGE-World`, we benchmark five models—GPT-4.1-mini [70], GPT-4o-mini [94], Llama-3-70B [39], Qwen-3-14B [118],

Task	Guidance	Human	ReAct					P&S	Reflexion
			4.1m	4om	Qwen14	Llama	Qwen235	4.1m	4.1m
Nav	Perfect	96%	85%	63%	58%	38%	58%	56%	86%
	Incorrect	72%	72%	19%	24%	12%	31%	25%	75%
	Misaligned	78%	65%	24%	28%	15%	31%	22%	67%
	None	44%	30%	32%	10%	1%	32%	10%	30%
Cook	Perfect	89%	98%	68%	88%	78%	93%	72%	98%
	Incomplete	78%	96%	48%	88%	76%	94%	50%	98%
	Misaligned	100%	91%	43%	46%	40%	72%	48%	94%
	Underspecified	100%	94%	48%	79%	77%	90%	60%	94%
Game	Perfect	61%	70%	46%	45%	44%	69%	56%	70%
	Incomplete	67%	48%	28%	17%	11%	32%	42%	50%
	Incorrect	44%	12%	16%	19%	22%	26%	10%	12%
	Misaligned	78%	42%	15%	28%	22%	42%	19%	44%
	Underspecified	83%	52%	22%	15%	2%	39%	34%	55%
	None	33%	16%	10%	1%	0%	8%	8%	18%

Table 3.2: MIRAGE-World results for ReAct, Plan & Solve (P&S), and Reflexion across different guidance. ReAct and P&S are evaluated for 1 trial and Reflexion for up to 3 trials. Numbers denote success rates (%) out of 100 tasks per condition, averaged across merged categories and rounded to the nearest integer. Abbreviations: **4.1m** = GPT-4.1-mini, **4om** = GPT-4o-mini, **Qwen14** = Qwen-3 14B, **Qwen235** = Qwen-3 235B, **Llama** = LLaMA-3 70B.

and Qwen-3-235B [118], each prompted with one-shot prompting. Agents receive structured natural language inputs that serialize the task description, goal, guidance, current observation, and episode history. For `MIRAGE-Web`, we evaluate GPT-4o on four WebArena domains: Reddit, Shopping, CMS, and GitLab with two-shot demonstrations. All agents interact with the environment through natural language actions. More details are in Appendix B.3.

3.4.1 MIRAGE-World Results

We evaluate agent performance under imperfect guidance using three representative baseline methods. (1) ReAct [170] combines chain-of-thought reasoning with action execution, prompting the LLM to interleave thoughts and actions in a step-by-step fashion. (2) We adopt a Plan-and-Solve (P&S) strategy with iterative decomposition [153, 56]. The agent incrementally generates and executes one plan step at a time with a separate ReAct agent. (3) Reflexion [132] allows agents to learn from prior failures by generating self-feedback after each episode. The agent iteratively updates its behavior by incorporating insights from

previous attempts. Table 3.2 reports task success rates under varying guidance conditions. We summarize several key findings:

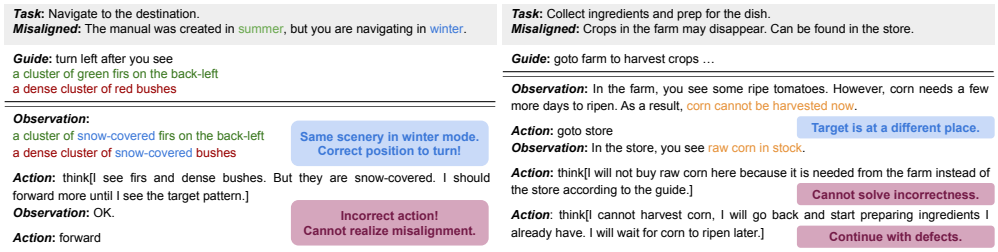


Figure 3.3: Two examples where the agent fails under imperfect guidance. Left: it does not recognize the misalignment. Right: it realizes the guidance is inapplicable but cannot recover and continues with flawed actions.

Perfect guidance boosts performance across all models and domains compared to the no-guidance baseline (i.e., one-shot ReAct).[1] While humans are generally robust to imperfect guidance and may even benefit from it by becoming more attentive, LLM agents typically suffer substantial performance drops regardless of the model or prompting framework. This reveals a core limitation of current LLM agents: while the tasks themselves remain solvable, agents struggle to detect flaws or adapt their behavior when the auxiliary guidance is inaccurate, outdated, or ambiguous. Figure 3.3 illustrates qualitative examples where agents either fail to detect imperfections in the guidance or are unable to recover effectively.

The underperformance of Plan-and-Solve (P&S) suggests that the challenge of MIRAGE-World does not lie in long-horizon reasoning, but in recovering from misleading or missing information. This is something that vanilla planning does not adequately address. Reflection-based methods like Reflexion perform slightly more robustly under imperfect guidance, confirming that agents benefit from reasoning over time. This also indicates that our environments provide cues, such as observable outcomes or intermediate feedback, for agents to learn and adapt. Nonetheless, our broader goal is to enable agents to succeed within a single episode, without relying on multiple retries.

The results also indicate that some imperfection types are inherently more difficult. *Incorrect* and *misaligned* guidance can be misleading, resulting in steepest performance drops, while agents handle *underspecified* and *incomplete* guidance better, likely due to broad pre-training knowledge. These findings validate the design of MIRAGE-World: it presents diverse yet solvable challenges that probe the guidance-following behavior of LLM agents.

[1]We omit the no-guidance condition for cooking tasks, as recipe information encodes critical task constraints. Without it, agents lack a meaningful basis for decision-making.

Domain	Perfect	Incomplete	Underspecified		Incorrect	Misaligned	None
		Skip Step	Narrative	Merged	UI Change	Obs Delay	
Reddit	86.67%	37.14%	35.23%	40.95%	56.19%	35.24%	11.43%
Shopping	62.14%	38.57%	44.29%	42.86%	53.57%	37.14%	32.09%
CMS	43.75%	18.75%	24.11%	18.75%	35.71%	18.75%	15.18%
GitLab	57.29%	36.46%	38.54%	35.42%	40.63%	30.21%	26.04%

Table 3.3: ReAct success rate with GPT-4o on WebArena tasks with imperfect guidance.

3.4.2 MIRAGE-Web Results

To assess execution-time robustness in a realistic setting, we evaluate GPT-4o on WebArena using both perfect and perturbed guidance derived from human demonstrations. Results are shown in Table 3.3. Although the "perfect" guidance provides one viable way to solve the tasks, tasks are not all successful due to limitations unrelated to guidance quality, such as difficulty interpreting tabular content or lack of persistent memory. Nevertheless, such guidance consistently leads to much higher performance than the no-guidance setting, confirming its utility. In contrast, all forms of perturbed guidance result in notable performance drops. Structural flaws such as skipped or merged steps disrupt execution flow, while underspecified narrative-style guidance degrades performance substantially (e.g., -51.44% on Reddit). These results highlight the sensitivity of current LLM agents to imperfect guidance, especially in long-horizon web tasks with verbose, interaction-heavy observations.

3.4.3 Additional Analyses

3.4.3.1 Imperfect-Aware Agent

We evaluate ReAct agents that are augmented with imperfection-aware prompting. This agent receives natural language hints describing the expected imperfection type (for example, *the direction may be misleading*) and a demonstration showing plausible recovery behavior. To mitigate the potential influence of the provided few-shot demonstrations, we selected 5 diverse examples and ran 1-shot experiments with each of the examples.

Results are shown in Table 3.4, **IA** denotes imperfection-aware prompting, where the agent is informed that the guidance may be flawed; **NA** indicates standard prompting, where no such awareness is provided. In scenarios where the imperfection is easier to identify and recover from, such as the storage loss setting in Cooking, the agent benefits from the imperfect-aware prompting, suggesting that simple priors can support more effective reasoning. However, in more challenging cases such as incorrect number of tiles in navigation, the agent shows no improvement. Moreover, performance under perfect guidance can

also degrade, as the additional hint causes the agent to second-guess accurate instructions and make unnecessary deviations. Overall, while imperfection-aware prompting can enhance robustness in certain contexts, it may also introduce new sources of mistakes.

	Navigation					Cooking	
Agent	**Correct Tiles**	**Wrong Tiles**	**Construct Tiles**	**Correct Obs**	**Seasonal Obs**	**Perfect**	**Storage Loss**
NA	90%	22%	8%	46%	33%	65%	48%
IA	90%	16%	6%	43%	30%	58%	60%

Table 3.4: Comparison between **NA** = Non-Aware and **IA** = Imperfect-Aware agents.

3.4.3.2 Guidance Imperfection Severity

To better understand how guidance quality affects agent performance, we design a controlled degradation experiment in the Reddit domain. Specifically, we randomly omit increasing proportions of steps from the demonstration steps: 0% (perfect), 25%, 50%, and 100% of steps skipped.

As shown in Figure 3.4, agent performance drops sharply as more steps are removed, demonstrating that degraded guidance often does more harm than good. In fact, when 50% of the steps are omitted, performance is sometimes worse than no-guidance, which suggests that partially flawed guidance can mislead the agent more than it helps. These findings show that current LLM agents heavily depend on explicit, step-level instructions and struggle with plans with skipped information that require inference or causal reasoning. Our results underscore the value of robustness benchmarks with graded imperfection levels, which help identify thresholds where model performance breaks down.

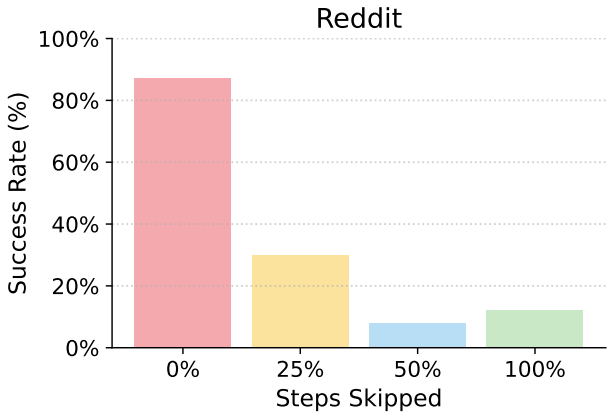


Figure 3.4: Effect of skipping steps in web guidance.

3.4.3.3 Error Analysis

Figure 3.5 shows common failure modes of GPT-4.1-mini ReAct agent in MIRAGE-World. Errors generally fall into four categories: (1) failing to notice the imperfection, (2) recognizing the issue but failing to recover, (3) not following correct guidance, and (4) general execution mistakes.

We observe that the dominant error types differ across tasks. For example, under guidance changes, navigation tasks often marked by the failure to notice flawed guidance, whereas gaming tasks more frequently involve being misguided by the changes. In contrast, when comparing navigation tasks under guidance versus environment changes, the former shows more instances of imperfection neglect, while the latter exhibits a higher rate of misguidedness. This variation highlights that current agents still struggle across multiple dimensions—including imperfection detection, reasoning, reflection, and robust execution.

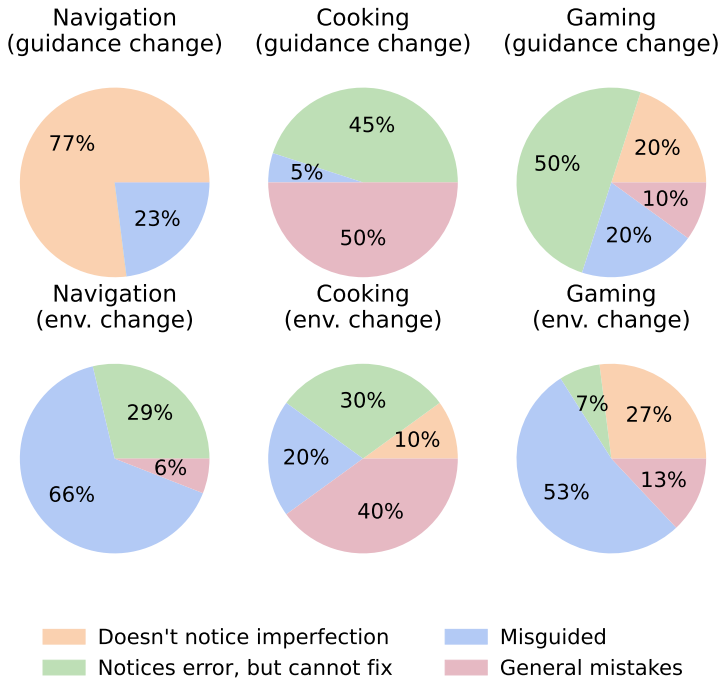


Figure 3.5: Error types of GPT-4.1-mini ReAct agents.

3.5 Conclusion

In this chapter, we introduce MIRAGE, a benchmark for evaluating the robustness of large language model agents under imperfect guidance. Our framework spans procedurally generated environments in navigation, cooking, and gaming, as well as realistic web-based tasks. By introducing guidance imperfections such as omissions, inaccuracies, and misalignments, we expose key limitations in current agents and highlight the need for adaptive behaviors like error detection and context-aware correction. Our evaluation shows that even modest deviations from ideal guidance can cause significant failures across common agent paradigms, highlighting the importance of developing more resilient and flexible agents. MIRAGE supports scalable task difficulty and will be released to facilitate research on robust decision-making in LLM agent design.

CHAPTER 4

Go Beyond Imagination: Maximizing Episodic Reachability with World Models

4.1 Introduction

Efficient exploration in state space is a fundamental challenge in reinforcement learning (RL) [49, 76], especially when the environment rewards are sparse [97, 96, 128] or absent [84, 107]. Such reward sparsity makes RL algorithms easy to fail due to the lack of useful signals for policy update [123, 29, 129]. A common approach for exploration is to introduce self-motivated intrinsic rewards such as state visitation counts [139, 68] and prediction errors [138, 108, 12]. Most of these intrinsic reward designs measure lifelong state novelty and prioritize visiting states that are less visited starting from the beginning of training.

While the above methods achieve great improvement on hard-exploration tasks like Montezuma's Revenge [12], they generally only work well on "singleton" environments, where training and evaluation environments are the same. However, due to the poor generalization performance of reinforcement learning in unseen environments, nowadays researchers have been paying more attention to procedurally generated environments [21, 20, 28], where the nature of task remains the same but the environment is randomly constructed for each new episode. For example, a maze-like environment will have different maze structures, making it rare for the agent to encounter the same observations across different episodes. Therefore, lifelong novelty intrinsic motivations usually fail in hard procedurally generated environments of this kind [120, 177] because an agent will be trapped around newly generated states.

Inspired by human's frequent use of short-term memory [4, 25] to avoid repeatedly visiting the same space, recent work proposes to derive intrinsic rewards on episodic level [126, 6, 120, 177, 181]. The episodic intrinsic rewards generally give a bonus to large episodic-level state space visitation coverage, therefore encourage visiting as many states as possible in the same episode. However, *does visiting more states necessarily mean efficient episodic-level exploration*? We notice that some state visitations are unnecessary and can be avoided if

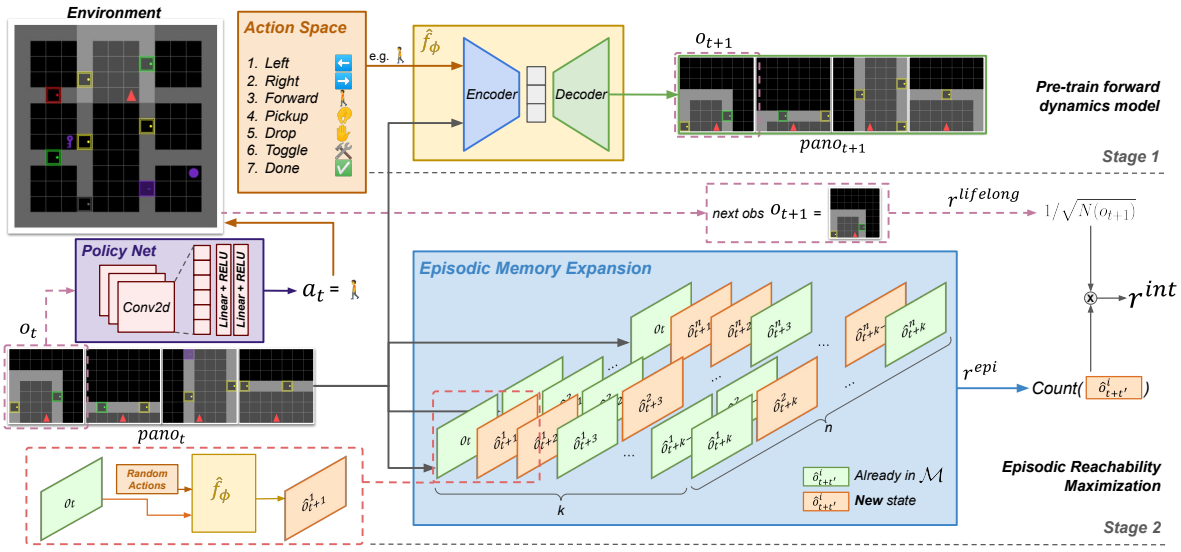


Figure 4.1: Illustration of how GoBI works on MiniGrid. For the environment in the upper-left corner, the red triangle indicates the position and orientation of the agent. It has a 7×7 partially-observable view (highlighted). During pre-training (Stage 1), we collect data using a random policy to train a forward dynamics model $\hat{f}_\phi(pano_t, a_t) = o_{t+1}$, where $pano_t$ denotes the panoramic view as defined in Section 4.3.1. For policy training (Stage 2), we apply $\hat{f}$ to predict observations in the future k timesteps with n random actions for each step. We add the new ones to an episodic buffer $\mathcal{M}$ and take the change of size of $\mathcal{M}$ as the episodic intrinsic reward r^{epi}. The lifelong intrinsic reward is COUNT-based. Our intrinsic reward GoBI is $r^{\text{lifelong}} * r^{\text{epi}}$.

they are predictable from episodic memory. For example, when navigating through a house to find a fridge, if you open a door and find an empty room, you do not need to go into it anymore because you can easily predict what the states are like in the room (i.e., intuitively speaking, you would be moving around in an empty room). With this inspiration, we propose to design the episodic intrinsic reward to not only maximize the number of visited states in an episode, but also consider those states that are not visited but can be predicted from episodic memory.

More precisely, we maintain an episodic buffer to store all the visited states as well as states reachable from the visited states within a few timesteps. To get the reachable states, we train a world model with forward dynamics function and apply random actions to the learned dynamics model to predict future states. The predictions are added to the episodic buffer if they are not there already. We use the change of size of this episodic buffer as the episodic intrinsic reward. Following prior work, we weight the episodic intrinsic reward by a lifelong intrinsic reward [6, 181] like the COUNT-based rewards. With this newly proposed intrinsic reward design **GoBI - Go Beyond Imagination**, the agent is expected to both

explore most of the state space throughout training to discover extrinsic rewards, and learn to act in an efficient manner within a single episode to avoid being trapped by seemingly novel states.

The contributions of this chapter can be highlighted as follows: (i) We propose a novel way to combine world models with episodic memory to formulate an effective episodic intrinsic reward design. (ii) In sparse-reward procedurally generated MiniGrid environments [17], GoBI greatly improves the training sample efficiency in comparison with prior state-of-the-art intrinsic reward functions. (iii) GoBI extends well to DeepMind Control Suite [147] with high-dimensional visual inputs and shows promising results on sparse-reward continuous control tasks. (iv) We analyze the design of GoBI and present extensive ablations to show the contribution of each component.

4.2 Method

We consider reinforcement learning problems framed as Markov Decision Process (MDP) $M = (\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state space and action space. $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function. γ is the reward discount factor. At each step t, the state of the environment is denoted as $s_t \in \mathcal{S}$. The agent generates an action $a_t \in \mathcal{A}$ to interact with the environment. The environment then transits to the next underlying state $s_{t+1} \in \mathcal{S}$. Apart from the new state s_{t+1}, the environment also returns an extrinsic reward r^{ext} that describes how well the agent reacts to s_t. In sparse-reward tasks, r^{ext} is usually 0. In this chapter, we follow the previous work to train RL algorithms with $r^{\text{ext}} + \lambda * r_t^{\text{int}}$, where r_t^{int} is a self-motivated intrinsic reward and λ is a hyper-parameter that controls the relative importance between intrinsic and extrinsic rewards.

4.2.1 Go Beyond Imagination

Reachable States and Episodic Buffer Our intrinsic reward design aims to exploit the information hidden inside the neighborhood of states. We define a state A to be k-step *reachable* from state B if the agent can reach A from B within k timesteps. During the training process, for each new episode, we initialize an empty episodic memory buffer $\mathcal{M}$. At timestep t, we hash s_t as well as all the states reachable from s_t. We denote the set containing all the hash codes of s_t and its reachable states as $\mathcal{M}_R(s_t)$. Then we update $\mathcal{M}$ by $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{M}_R(s_t)$. Storing the hash codes instead of directly storing the states may alleviate the potential memory issue of the buffer. We illustrate this process in Figure 4.2.

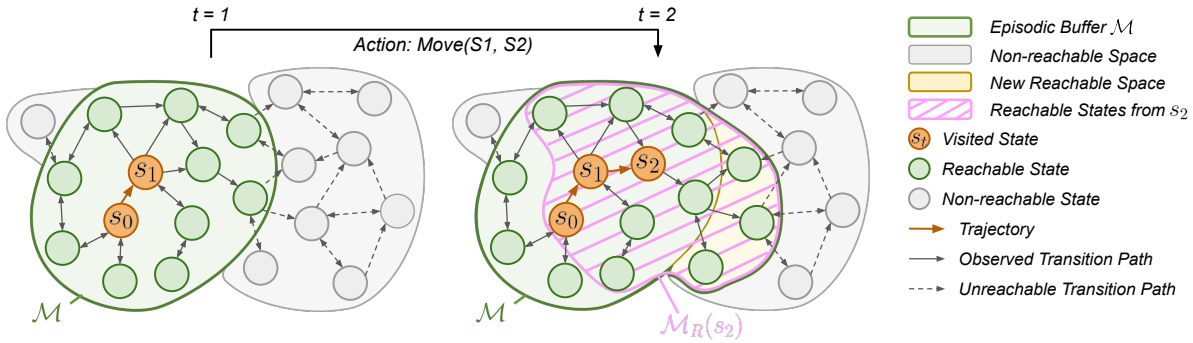


Figure 4.2: An illustration of how our episodic buffer updates. We consider reachable states that are $k = 2$ timesteps away from s_t. After the agent moves from s_1 to s_2, the episodic buffer $\mathcal{M}$ (shaded in green) expands by 3 new reachable states (shaded in yellow). More formally, $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{M}_R(s_2)$, where $\mathcal{M}_R(s_2)$ indicates a set of all the states 2-step reachable from s_2. Notice that all the states in the trajectory, i.e., s_0, s_1, s_2 are also added to the buffer.

When the agent reaches state s_2, we add 3 more states that are reachable from s_2 but not in $\mathcal{M}$.

Forward Dynamics In real environments, it is common that we do not have access to the neighborhood relationship between states. However, we can learn a world model by training a forward dynamics model $\hat{f}_\phi(s_t, a_t) = s_{t+1}$ to predict the states reachable from s_t. This forward dynamics can be pre-trained using data collected by a random policy or trained online together with policy training. When training the policy, for each timestep t, we generate $k \cdot n$ random actions and use the learned dynamics $\hat{f}_\phi$ to predict states in the future k steps. We hash the current state s_t as well as the predicted future states $\hat{s}^1_{t+1}, ..., \hat{s}^n_{t+1}, ..., \hat{s}^1_{t+k}, ..., \hat{s}^n_{t+k}$ and add the hash codes to the episodic buffer $\mathcal{M}$ if they are not in the buffer. Apart from alleviating potential memory issue as mentioned in the last paragraph, using a hash function may also mitigate the noise introduced by $\hat{f}$. With a learned dynamics model, the predictions of reachable states are usually not perfect. However, in the experiment section we show that even with imperfect predictions, our method can improve the training sample efficiency a lot.

Episodic Novelty We aim to design an episodic-level novelty reward that guides the agent to extend the frontier of its predicted reachable space efficiently to discover states not visited and not predictable within the same episode. More specifically, we denote the size of the episodic buffer $\mathcal{M}$ as m_t at timestep t and design a reachability-based bonus $r^{epi} = m_{t+1} - m_t$ that encourages the agent to find unexplored regions. For each timestep, the agent is expected to reach the state that is reachable to more new states in the current

Algorithm 3 Go Beyond Imagination

Input: Intrinsic Reward Coefficient λ_0, Forward Prediction Step k, Number of Random Actions n, Intrinsic Reward Decay Parameter ρ
Initialize policy π_θ, dynamics model $\hat{f}_\phi$, replay buffer $\mathcal{B}$.
(Optional) Collect episodes with π_θ and train $\hat{f}_\phi$ with prediction loss
for episode $e = 1, 2, ...$ until convergence **do**
  Initialize episodic buffer $\mathcal{M}$.
  $\lambda \leftarrow \lambda_0 * (1-\rho)^{(e-1)*T}$
  for $t = 1$ **to** T **do**
    Execute π_θ in the environment to get a transition pair $(s_t, a_t, s_{t+1}, r_t^{\text{ext}})$.
    $m_t \leftarrow size(\mathcal{M})$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{hash(s_t)\})$
    for $t' = 1$ **to** k **do**
      generate n random actions $a_{t'}^1, ..., a_{t'}^n$
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{hash(\hat{f}_\phi(\widehat{s}_{t+t'}^i, a_{t'}^i) | i = 1, ..., n\})$
    end for
    $r_t^{\text{int}} = (size(\mathcal{M}) - m_t) * r^{\text{lifelong}}$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, s_{t+1}, r_t^{\text{ext}} + \lambda * r_t^{\text{int}})\}$
  end for
  update π_θ with RL objective
  update $\hat{f}_\phi$ with prediction loss
end for

episode.

Intrinsic Reward Formulation We further weight our episodic intrinsic reward by a lifelong intrinsic reward to encourage the agent to explore the regions that are not well explored in the past. More formally, the proposed intrinsic reward GoBI is defined as:

$$r_t^{\text{int}} = (m_{t+1} - m_t) * r_t^{\text{lifelong}} \tag{4.1}$$

Here, r_t^{lifelong} denotes lifelong intrinsic reward. We note that our framework is compatible with any choice of lifelong intrinsic reward. Specifically, we use the simple COUNT-based reward $1/\sqrt{N(s_{t+1})}$ for the navigation experiments on MiniGrid environments [17], where N denotes the count of s_{t+1} from the start of training.[1] For the experiments on DeepMind Control Suite [147] we use the state-of-the-art intrinsic reward RE3 [130], which estimates state entropy by a random encoder.

Intrinsic Decay Intrinsic rewards are expected to be asymptotically consistent so that it will not influence the policy learning at later stage of training and result in a sub-optimal

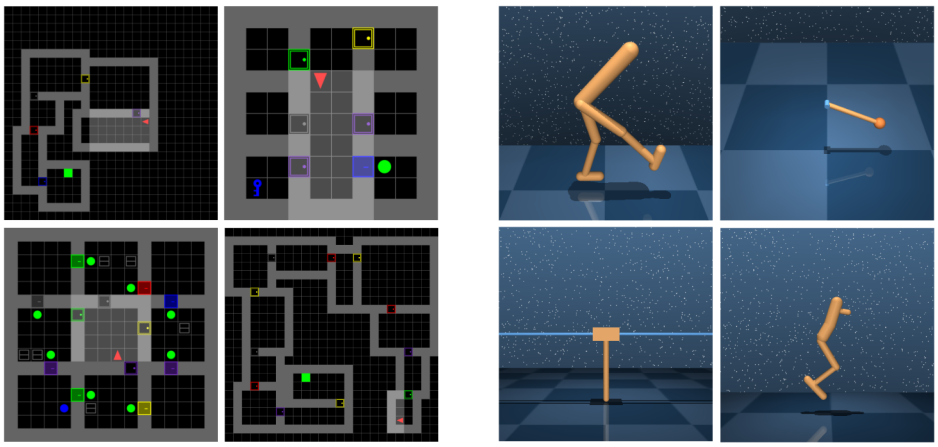
[1]For environments that are partially observable (e.g., in MiniGrid, the agent observes a 7×7 pixel local view of the environment), we substitute state s_t with observation o_t when calculating the intrinsic rewards.

policy. To guarantee that the policy learning focuses more on extrinsic rewards as training proceeds, in RE3 [130], the authors apply exponential decay schedule for the intrinsic rewards to decrease over time. Although COUNT-based reward theoretically converges to 0 with enough exploration, it decreases quite slowly in procedurally generated environments. Therefore, we also apply intrinsic reward decay when calculating GoBI by decreasing the intrinsic reward coefficient λ during training. We summarize our method in Algorithm 3 and illustrate the training process on MiniGrid navigation tasks in Figure 4.1.

4.2.2 Conceptual Advantage of GoBI over Prior Works

Previous works including RIDE [120] and NovelD [181] also combine episodic intrinsic reward with lifelong novelty as we do. However, most of them focus on episodic-level state visitation. For example, NovelD only assigns non-zero rewards to a state when it is visited for the first time in the episode. However, we notice that not all state visitations are necessary. The agent's goal for exploration is to gather information about the states. Therefore for states that are easily predictable from episodic memory, visiting them may not really help to acquire more information about the environment. In Figure 4.4, we plot the visitation heatmap of GoBI and NovelD to demonstrate the different exploration behaviors of the two methods.

Our method is closely related to another work that measures episodic curiosity (EC) [126]. In EC, the authors train a reachability network that takes in two arbitrary states and outputs a similarity score between 0 and 1, where 1 indicates the two states are the same and 0 indicates they are totally different. The network is trained using collected episodes by marking temporally close states as positive examples and temporally far ones as negative samples. Meanwhile, they also maintain an episodic buffer. A state s_t is compared with all the states in the buffer and gets a high intrinsic reward if the corresponding similarity scores are low. Only with low enough similarity scores do they add s_t to the buffer. Although their method and ours are similar at high level, they are different by design. For example, for an agent standing in front of an empty blind alley with dead end, agent trained with GoBI does not benefit from going deep into the blind alley because everything there can be predicted as reachable and added to the episodic buffer already. However, EC encourages going to the very end of the blind alley to reach the state with low similarity score and high intrinsic reward, even though going into an empty blind alley is not beneficial for exploration and wastes time that can be used to explore other parts of the environment. In Appendix C.3, we present the visitation heatmaps of policies learned by EC and find that it prefers going to the room corners, which well matches our explanation above.



(a) MiniGrid (b) DeepMind Control

Figure 4.3: Rendering of the environments used in this chapter. Left: 2D grid world navigation tasks that require object interactions. Right: DeepMind Control tasks with visual observations.

4.3 Experiments

In this section, we evaluate GoBI in two domains: 2D procedurally generated MiniGrid environments [18] with hard-exploration tasks and locomotion tasks from DeepMind Control Suite [147]. The experiments are designed to answer the following research questions: (1) How does GoBI perform against previous state-of-the-art intrinsic reward designs in terms of training-time sample efficiency on challenging procedurally generated environments? (2) Can GoBI successfully extend to complex continuous domains with high-dimensional observations, for example control tasks with visual observations? (3) How does each component of our intrinsic reward contribute to the performance? (4) What is the influence of the accuracy of the learned world models to our method?

4.3.1 MiniGrid Navigation Tasks

MiniGrid Environments MiniGrid [18] is a set of partially-observable procedurally generated grid world navigation tasks. The agent is expected to interact with objects such as keys, balls, doors, and boxes to navigate through rooms and find the goal that is randomly placed in one of the rooms. The tasks only provide one sparse reward at the end of each episode, which indicates if the agent successfully finds the goal or not and how many steps it takes to reach the goal. In this chapter, we consider 3 types of tasks including MultiRoom, KeyCorridor, and ObstructedMaze. Some environments that we experiment on in this chapter are shown in Figure 4.3a. The upper-right is a KeyCorridorS4R3 environment, where the

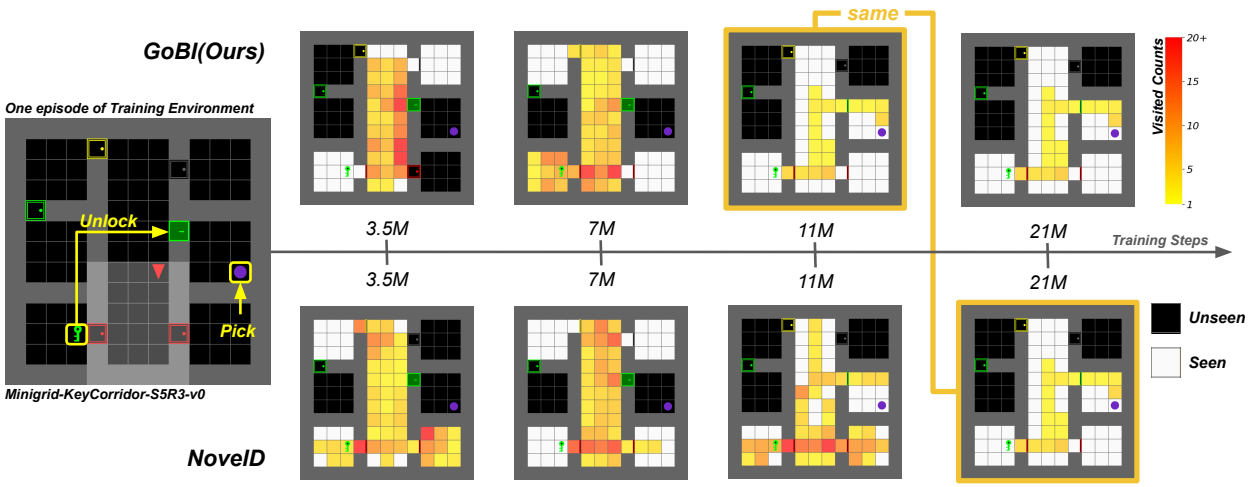


Figure 4.4: Visitation heatmaps on KeyCorridorS5R3 at different training stages. This figure compares the policy behavior of GoBI and NovelD. A dark red color means plentiful visitations, white means the agent has seen the space but did not step on it, and black indicates spaces that are not discovered. It is worth noticing that early in the training (3.5M and 7M timesteps), our policy already learns not to go into an empty room, likely because states in an empty room are easily predictable. On the contrary, even after 11M steps, an agent trained with NovelD still goes into an empty room (bottom-right corner) for more state visitations.

agent should learn to open the doors to find a key, use it to open the locked blue door, and pick up the green ball. The bottom-left figure shows an ObstructedMaze-Full environment, which is similar to KeyCorridor but more challenging. The rooms are larger, the doors are blocked by balls, and the keys are hidden in boxes. The upper-left and bottom-right environments are MultiRoom environments, in which the agent has to navigate through connected rooms to reach the goal in the last room.

Baselines We compare with state-of-the-art intrinsic reward designs that work well on MiniGrid including NovelD [181], RIDE [120], and RND [12]. For a fair comparison, we follow the same basic RL algorithm and network architectures used in the official codebase of NovelD and only change the intrinsic rewards r^{int} for all the methods. We also compare our method with EC [126] because of the similarity of the high-level idea between the two methods. However, the original paper of EC does not include experiments on MiniGrid. Therefore, we implement our own version to adapt to MiniGrid. We follow their implementation suggestions and tune the hyper-parameters such as novelty threshold by grid search.

Dynamics Model Training For each experiment on MiniGrid, we first run a random policy for 1e5 steps to collect data and use them to train a forward dynamics model as

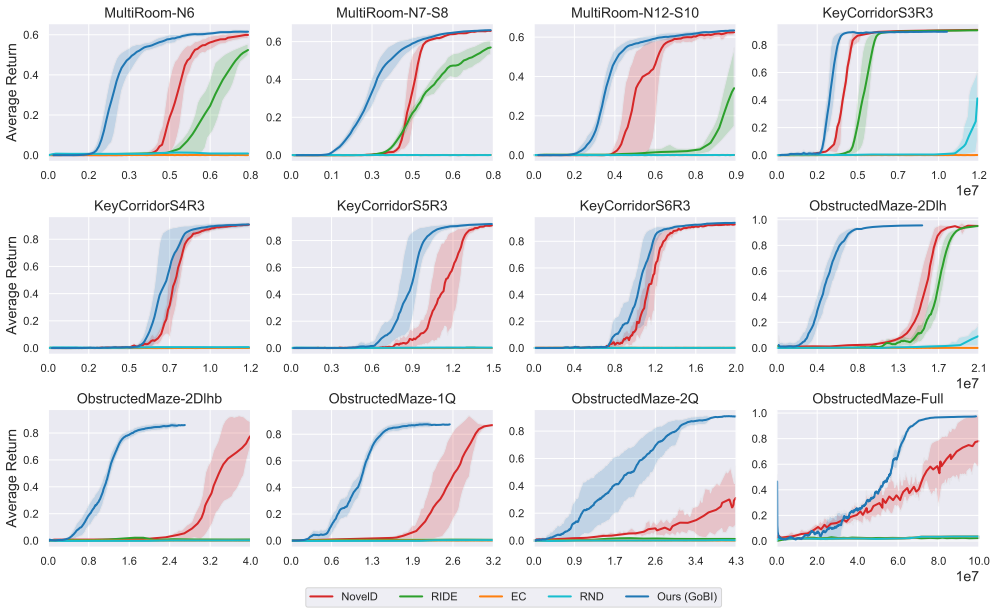


Figure 4.5: Training performance of GoBI and the baselines on 12 MiniGrid environments. The x-axis shows the number of environment steps. We shift the training curves towards right by the number of environment steps we use to pre-train the dynamics model, i.e., 1e5 timesteps. Results are averaged across 4 seeds.

the world model. Among the pairs collected, there are about 5e4 different transition pairs. During our experiments, we observe that fine-tuning the pre-trained dynamics model during policy training has no significant influence on the performance. Similar to [107], we use the 360° panoramic views as the input to predict the future observations. This is a rotation-invariant representation of the observed state. We consider this still a fair comparison with the previous state-of-the-arts because both NovelD and RIDE rely on using the state information instead of observations for the episodic count calculation.

Due to the limited field of view of the agent, we only forward the learned dynamics by $k = 1$ step when predicting. We predict the next observations produced by all 7 discrete actions in the MiniGrid tasks including turn left, turn right, forward, toggle, pick up, drop, and done. We directly apply the default Python hashing function to hash the observations and predicted future observations. We do not expect the hashing function to mitigate the prediction error on MiniGrid, but only use it to reduce the dimension of observations and predictions.

Training Performance on MiniGrid Figure 4.5 shows the learning curves of GoBI and state-of-the-art exploration baselines NovelD, RIDE, RND, and EC on 12 most challenging MiniGrid navigation tasks, including MultiRoom, KeyCorridor, and ObstructedMaze. Our

curves are shifted towards right by the number of random exploration environment steps used to train the world model. In all 12 environments, GoBI significantly outperforms previous methods in terms of sample efficiency. For instance, on ObstructedMaze-2Dlhb, GoBI is about three times more sample efficient than NovelD. On the hardest ObstructedMaze-Full environment, GoBI achieves near-optimal performance within 70M steps. Lastly, although we try to tune the hyper-parameters of EC, our implementation of EC still does not learn well on the MiniGrid environments.

Qualitative Results To clearly present the exploration behavior learned by GoBI, we show the visitation heatmaps of GoBI and NovelD on a KeyCorridorS5R3 environment in Figure 4.4. Not only does our method converge to an optimal policy faster, the exploration behavior is very different from NovelD. GoBI quickly learns not to visit easily predictable states like an empty room, making it more efficient to explore interesting parts of the environment, for example, the room with a key in it.

4.3.2 Experiments on Control Tasks

We further test GoBI on DeepMind Control Suite, which are a set of image-based continuous control tasks. These tasks are more challenging than MiniGrid because of its high-dimensional observations and stochastic transitions. Notice that these environments are not procedurally generated. The experiments in this section are to show the generality of our method by experimentally showing that GoBI extends well to sparse-reward tasks with continuous action space and high-dimensional observation space.

Dynamics Model Training We follow the world model structure in Dreamer [46] and directly apply their encoder, transition model, and observation model to predict future observations. However, compared to MiniGrid, it requires way more data to train a decent dynamics model on DeepMind Control to generate visually-reasonable predictions. Therefore, different from the experiments on MiniGrid, we do not pre-train the dynamics models. Instead we train the dynamics model together with the policy as shown in Algorithm 3. We find that the number of sampled random actions $n = 5$ works well across all 4 environments. For the number of forward prediction steps k, we set it to be 3 for Pendulum Swingup and 1 for the other 3 environments.

For the hashing function, we find that a simple SimHash as suggested in [146] works well in capturing the similarities between similar observations. We use SimHash to hash the image observations to 50 bits.

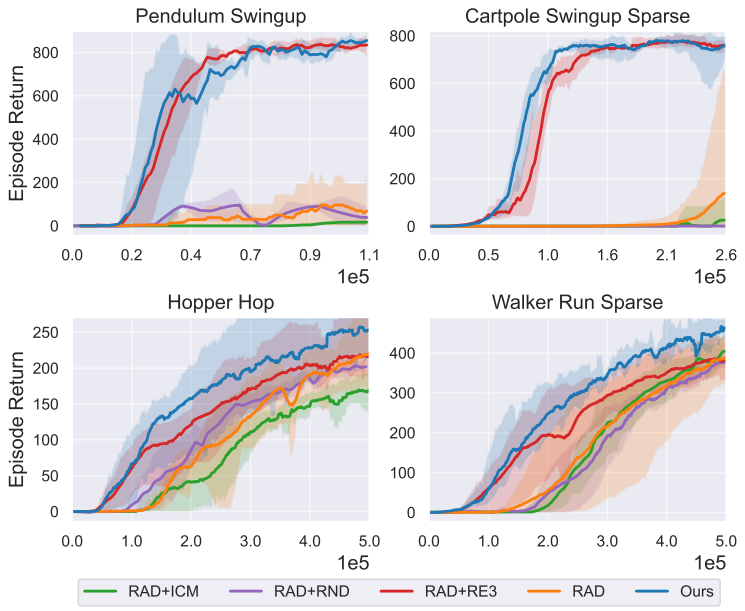


Figure 4.6: Training curves of GoBI and the baselines on DeepMind Control Suite. The curves are averaged across 5 seeds.

Training Performance on DeepMind Control We compare with the state-of-the-art intrinsic motivation on DeepMind Control tasks - RE3 [130], which applies a k-nearest neighbor entropy estimator in the low-dimensional representation space of a randomly initialized encoder to maximize state entropy. RE3 is also what we use for the lifelong intrinsic reward part r^{lifelong} of GoBI in Equation 4.1. Another two intrinsic reward baselines we consider are ICM [108] and RND [12]. For a fair comparison, all the experiments use the same basic RL algorithm RAD [74]. The results are shown in Figure 4.6. The additional episodic-level intrinsic reward term improves the sample efficiency a lot compared to only using lifelong intrinsic reward, especially on Hopper Hop and Walker Run Sparse.

4.3.3 Ablation Study

GoBI Variations In this section, we analyze how each component of our intrinsic reward contributes to the final performance. We ablate each component of GoBI and run experiments on MiniGrid environments with the following:

- R1: only episodic intrinsic reward $m_{t+1} - m_t$
- R2: indicator of whether the new states are added to the episodic buffer

$$(\mathbb{1}\{m_{t+1} - m_t > 0\})/\sqrt{N(o_{t+1})}$$

- R3: only lifelong intrinsic reward $1/\sqrt{N(o_{t+1})}$

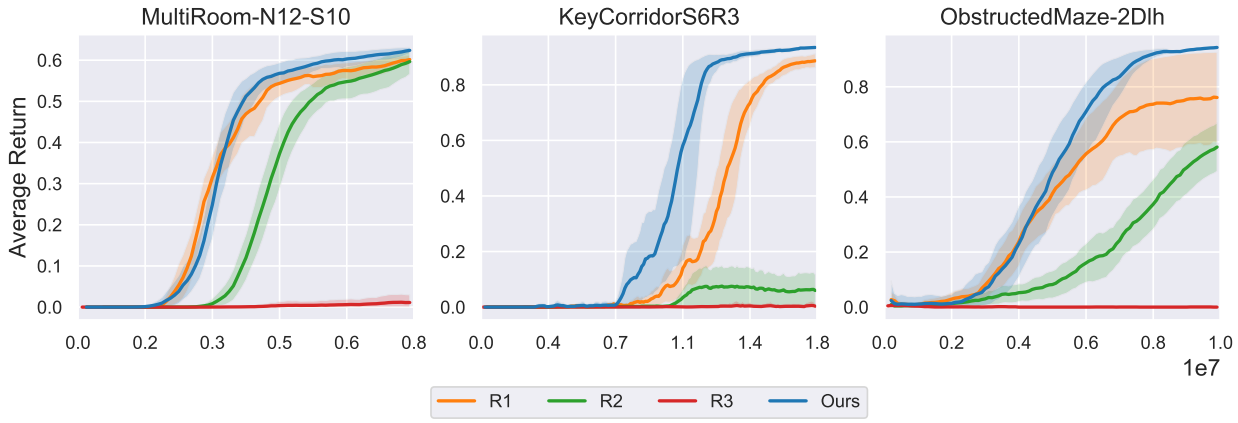


Figure 4.7: Training performance comparison among GoBI, R1, R2, and R3 on 3 MiniGrid environments.

Training performance of GoBI as well as R1, R2, and R3 are shown in Figure 4.7. Although R1 works on MultiRoom, it suffers on Obstructed Maze and large KeyCorridor environments. The underlying reason may be that in Key Corridor and Obstructed Maze the room structures change less across episodes than MultiRoom (all generated rooms are squares with fixed sizes), therefore the COUNT-based rewards contribute more in such environments than in MultiRoom. At the same time, R2 performs way worse than GoBI. Agents trained with R2 prefer actions that only increase the size of the episodic buffer a bit therefore getting positive score more often. We provide an illustrative example in Appendix C.6 to explain why R2 does not work well compared to GoBI. Using only lifelong intrinsic reward R3 performs the worst and struggles to learn efficiently on large MultiRoom, Key Corridor, and Obstructed Maze environments.

Real Dynamics vs Learned Dynamics A learned dynamics model is generally not perfect, especially for partially-observable environments like MiniGrid. In many cases the predictions can never be accurate. For example, when the agent first opens the door of a new room, usually it will not accurately predict everything behind the door. Figure 4.8 shows the training curves between using the real dynamics model vs a learned dynamics model. Not surprisingly, with the same intrinsic reward function, using the real dynamics converges faster to a near-optimal policy. However, even with imperfect dynamics model, our method still greatly surpasses previous state-of-the-art methods.

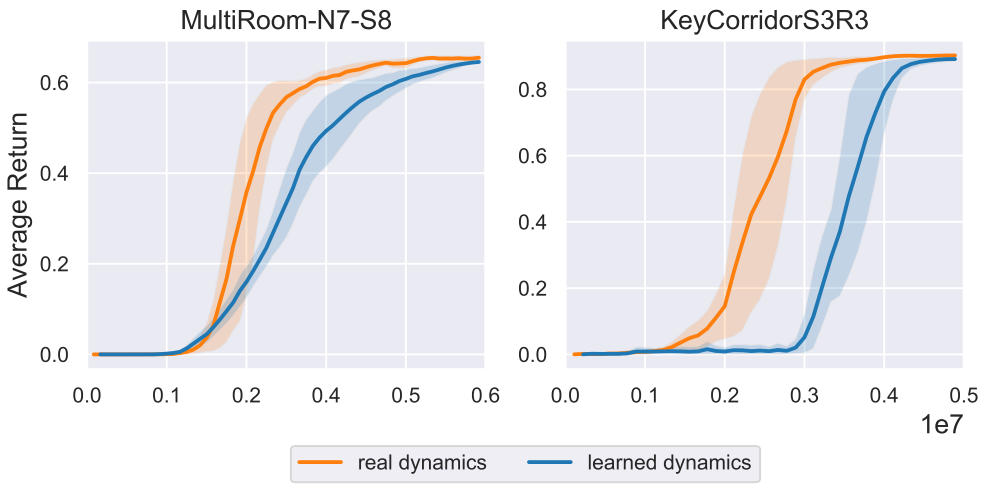


Figure 4.8: Comparison between using the real dynamics model of the environments vs using a learned one on MiniGrid environments. In both MultiRoom and KeyCorridor, using a real dynamics model to derive intrinsic reward makes the policy converge faster, especially on KeyCorridor.

Multi-Step Predictions Figure 4.9 shows the learning performance of GoBI on MiniGrid with varying choices of the number of future steps to do predictions $k = 1, 2, 3$. For $k > 1$, our dynamics model outputs $pano_{t+1}$ instead of o_{t+1} and we hash and store the observations from panoramas in each future timestep. With a real forward dynamics model, a larger k generally accelerates exploration more, because it prioritizes actions that lead to the states that are reachable to more states in the long run. However, due to the limited field of view of the agent and the model inaccuracy, this is not the case if we use a learned model. Forwarding 2 steps is still faster than only 1 step, but more steps than that does not really make exploration faster.

4.4 Related Work

4.4.1 Exploration in Reinforcement Learning

Efficient exploration in reinforcement learning, especially for sparse-reward reinforcement learning problems is challenging. A natural and popular solution is to design some metric to evaluate state novelty and assign high intrinsic reward to novel states. For example, COUNT-based intrinsic reward [139, 68, 146] and curiosity-based intrinsic motivation [138, 108, 12]. Another popular way is to do state space entropy maximization [49, 76]. Recently, nearest neighbor entropy estimation methods [171, 84] have shown great performance improvements in challenging visual domains. Our method is compatible with all these successful exploration

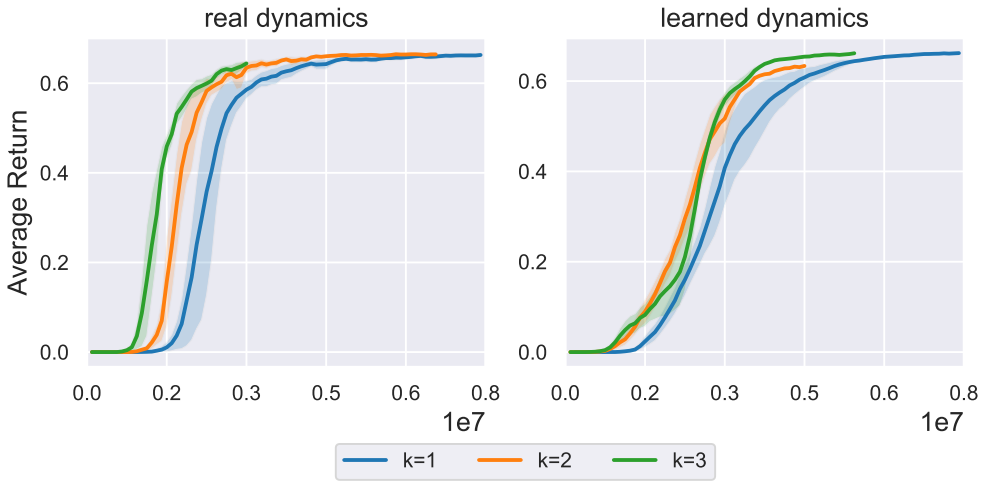


Figure 4.9: We make forward predictions for different numbers of future steps k using both the real dynamics and the learned dynamics model. The plots above show the training performance on MiniGrid MultiRoom-N7-S8.

intrinsic reward designs by using them as r^{lifelong}, but we additionally encourage the episodic-level reachable space expansion to achieve large state space coverage within a single episode.

4.4.2 Episodic Memory

Deriving useful information from episodic buffer have shown great success in improving the training sample efficiency in RL on navigation, control, and Atari games. Episodic memory buffers are applied to mimic hippocampal episodic control and rapidly assimilate recent experience [8, 114]. As is mentioned in the previous sections, [126] keeps an episodic buffer to store observations and introduce an episodic curiosity module to determine if a new observation is reachable from previous observations or not. RAPID [177] proposes a novel way to do behavior cloning on episodes with high episodic coverage. NGU [6] combines an episodic novelty module and a lifelong novelty module to generate intrinsic rewards. However, in NGU, the episodic novelty is a measurement of difference between the current observations from the previous observations, while ours focus on how much the reachable space is expanded from the new state. RIDE [120] and NovelD [181] both count the episodic state visitations, while we claim that apart from visited states, we should also consider states that can be predicted from short-term episodic memory.

4.4.3 Learning World Models with Forward Dynamics

Learning dynamics function from a set of observed data is a widely-studied topic in reinforcement learning, especially due to the rapid growth of model-based reinforcement learning

[155]. Existing work shows that an agent's world model is implicitly a forward model that predict future states [43, 30]. Recently, people have proposed latent dynamics models that work well on high-dimensional inputs [100]. These latent dynamics models encode image observations and predict future states in the latent space [44, 46, 47], outputting realistic future observations on visually complex domains including DeepMind Control Suite [147], VizDoom [63], Atari Games, and DeepMind Lab [7]. The learned dynamics models can be used to guide exploration by prediction error [138, 108, 12], surprise [2], or information gain by variance of model ensemble means [129]. Our method differ from the previous methods by directly generating and hashing the predicted states and add them to an episodic reachable state buffer. With the advanced world model structures, our method can be extended to diverse domains with complex observations.

4.5 Discussion: Test-Time Exploration of LLM Agents

Procedurally generated environments such as MiniGrid [18] are challenging because an agent cannot succeed merely by memorizing action sequences and replaying them at test time. In MiniGrid environments such as MultiRoom and ObstructedMaze, agents must learn efficient exploration strategies that guide them toward effective exploratory actions, such as opening boxes and doors to obtain clues about how to proceed.

Similarly, LLM-based agents are increasingly expected to handle a broad range of general-purpose tasks. This introduces challenges analogous to those encountered in procedurally generated environments. In general-purpose agent settings, test-time workflows may differ from the knowledge encoded in the model. Under such conditions, agent performance is often less robust than expected, as demonstrated in Chapter 3 and in prior work [16, 64].

One potential source of failure is that LLM-based agents are typically instructed or optimized solely to maximize task progress. However, in unfamiliar situations, effective exploratory actions can help gather task-relevant information for better decision-making, even though such actions may not directly contribute to task progress. For example, in a task such as buying a bottle of water, the most obvious milestone steps are to go to a store, find water, and purchase it. Actions such as asking a disambiguating question like *What brand do you want?*, checking a map, or asking someone for directions are all potentially helpful behaviors that do not directly contribute to task completion. An agent cannot be expected to have memorized the route to a store on every possible map. Thus, in such cases, temporarily adjusting the objective to efficient task-specific exploration could ultimately improve task completion.

4.5.1 Method

Motivated by this observation and inspired by GoBI, we conduct a simple experiment to examine whether explicit test-time exploration can benefit LLM-based agents. Given its conceptual simplicity and its use in prior work on LLM reasoning and agent search [187, 67, 81], we adopt a Best-of-N baseline. At each timestep t, the agent proposes N progress-oriented candidate actions, denoted by $\mathcal{A}_t^{\text{prog}}$, and an LLM-based evaluator ranks these candidates according to their expected contribution to task completion. The agent then executes the highest-ranked action.

We additionally introduce a two-stage action-selection procedure that explicitly distinguishes between progress-oriented and exploration-oriented behavior. As in the Best-of-N baseline, the agent first proposes and ranks a set of progress-oriented actions. However, the evaluator is also allowed to decide that the agent should explore rather than immediately execute one of the proposed actions.

When exploration is selected, the agent proposes a new set of N exploration-oriented candidate actions, denoted by $\mathcal{A}_t^{\text{explore}}$. These candidates are generated with a prompt that encourages task-relevant information-gathering behavior rather than undirected exploration. A second LLM-based evaluator then ranks the exploration candidates according to their expected usefulness for finding task-relevant information, and the agent executes the highest-ranked exploration action.

4.5.2 Preliminary Study

We evaluate the proposed exploration mechanism in a preliminary study using two MIRAGE [35] Roguelike environment variants in which the provided task guidance is misaligned with the test-time environment. Successful behavior therefore requires the agent to recognize that the provided guidance is unreliable and to explore the environment in order to identify an alternative strategy.

Environment	**Progress Only**	**Progress + Exploration**
Roguelike-Rename	12.5	**38.0**
Roguelike-Shuffle-Complex	25.0	**31.0**

Table 4.1: Preliminary comparison of success rates (%) on two MIRAGE-Misaligned Roguelike variants. The backbone model is `Qwen3.5-35B-A3B`.

As shown in Table 4.1, the proposed exploration mechanism improves performance on both Roguelike variants. These results suggest that explicitly switching from progress-

oriented action selection to information-gathering behavior when necessary can help LLM agents recover when the prior knowledge (guidance) is slightly off. These preliminary results demonstrate the promise of test-time exploration and motivate further research along this direction.

4.6 Conclusion

In this chapter, we introduce Go Beyond Imagination - GoBI, a novel episodic intrinsic reward design that encourages efficient episodic-level exploration by expanding reachable space. While most previous episodic intrinsic rewards use a naive episodic state count or state visitation coverage, our method exploits learned world models to predict reachable states and motivates the agent to seek for the states with more unexplored neighbors. Combined with lifelong intrinsic rewards, our method shows great training time sample efficiency improvement on hard procedurally generated environments. At the same time, it can be extended to guide exploration on continuous control tasks with visual inputs, both indicating a promising future in this direction.

CHAPTER 5

Hierarchical Procedural Meta-Reasoning for Generalizable Multimodal Agents

5.1 Introduction

There has been growing interest in building multimodal agents due to their ability to map rich multimodal observations to low-level actions in interactive environments [86, 185, 36]. Although recent foundation models fine-tuned on task-specific data have substantially improved perception and action generation [36, 24], achieving robust generalization in such settings remains an open challenge. In particular, a major factor limiting generalization is the absence of sufficient task-level knowledge that specifies how a goal should be accomplished under novel conditions [3, 133].

In this chapter, we study this challenge in the context of Android mobile navigation, which serves as a particularly suitable testbed for investigating generalization. Real-world mobile applications are highly diverse and exhibit rapidly evolving interfaces, user-specific configurations, and time-sensitive workflows. As a result, despite the availability of large-scale, high-quality Android interaction datasets [121, 143, 78, 13], agents fine-tuned for Android execution continue to struggle to generalize to realistic mobile usage scenarios [143].

A key limitation of existing mobile agents is their reliance on implicitly encoded task knowledge. Agents trained primarily from interaction trajectories excel at grounded actions, but tend to entangle task understanding with application-specific action patterns observed during training. This coupling makes them brittle under distribution shifts, as the agent lacks an explicit representation of how a goal should be achieved beyond the training distribution. Therefore, we consider task procedures as an explicit form of task knowledge, consisting of temporally abstract and human-interpretable steps that describe how a goal can be accomplished. Conditioning low-level action generation on such procedures provides global task structure for grounded execution. Indeed, we find that procedures generated by large language models already constitute a strong source of knowledge at test time and

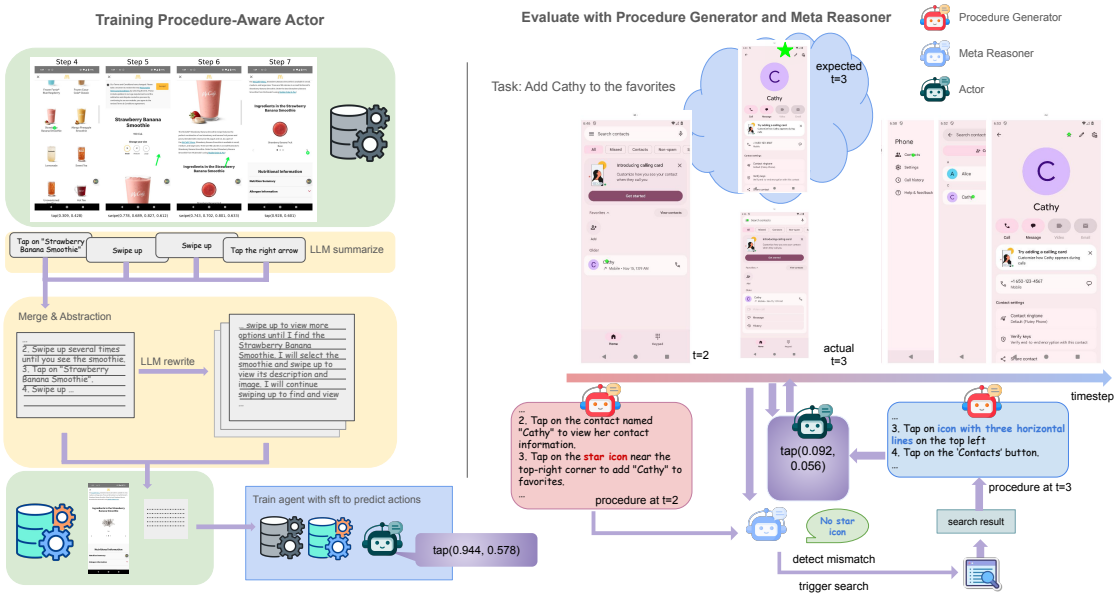


Figure 5.1: Overview of **P**rocedure-**A**ware **M**ultimodal **A**gent with Meta-**R**easoning (PAMAR). Left: Training. We train a procedure-aware grounded actor using supervised fine-tuning on a mixture of interaction trajectories augmented with induced natural-language procedures and their paraphrases, enabling the actor to reliably condition low-level actions on task-level procedural guidance. Right: Test time. The procedural generator produces and updates a task-level procedure online based on the current goal, observations, and interaction history. A meta-reasoning module evaluates whether the maintained procedure remains sufficient to explain ongoing execution; when a procedural deficiency is detected, external information retrieval is selectively triggered to help refine the procedure. Otherwise, execution proceeds using the internally updated procedure. [1]

substantially improve generalization.

However, procedural knowledge inferred by language models is not always sufficient. In realistic mobile settings, it can be outdated or mismatched, particularly for scenarios that postdate the training data of the LLM. Blindly following an incorrect procedure can therefore lead to systematic failure [145, 156] rather than graceful recovery. We may always invoke external information sources, such as web search or user clarification, whenever uncertainty arises. However, unconditional retrieval is fundamentally misaligned with interactive mobile agents for several reasons. First, external search interrupts closed-loop perception and action, introducing latency and breaking temporal coherence during execution. Second, retrieved information may be noisy, overly specific, or incompatible with the current interface state, potentially overriding correct internal knowledge with brittle instructions. Moreover, always-search policies eliminate the need for the agent to reason about the limits of its own

[1]The robot icon is made by Freepik from Flaticon.com, and the dataset icon is made by bsd from Flaticon.com. The windows search icon is provided by Icons8.

procedural understanding. By collapsing all uncertainty into retrieval, the agent never explicitly distinguishes between cases where its existing knowledge is sufficient and cases where it is not.

This motivates a central question: can an agent explicitly assess whether its current procedural knowledge is *sufficient to reliably complete a task*? To this end, we propose **P**rocedure-**A**ware **M**ultimodal **A**gent with Meta-**R**easoning (PAMAR), a procedure-aware multimodal agent that explicitly reasons about the sufficiency of its task-level knowledge during execution. PAMAR represents task understanding as natural-language procedures and decouples high-level procedural reasoning from low-level grounded action generation. Crucially, the agent is equipped with a meta-reasoning mechanism that continuously evaluates whether its current procedure remains adequate given the observed interaction context, and selectively triggers external information acquisition only when procedural deficiencies are detected. This design enables the agent to leverage internal procedural knowledge when it is sufficient, while robustly incorporating external information when it is not, leading to improved generalization under distribution shifts.

Our contributions are summarized as follows:

- We introduce *procedural sufficiency* as a key abstraction for multimodal agents, and propose a meta-reasoning mechanism that explicitly evaluates whether the agent's current procedural knowledge is adequate for task completion, enabling selective external information acquisition.
- We propose an LLM-guided training framework that induces temporally abstract natural-language procedures from demonstrations to guide low-level action generation, decoupling high-level procedural reasoning from grounded execution to improve generalization.
- Our proposed method demonstrates substantial generalization improvements on challenging Android benchmarks, including unseen applications, post-training interface changes, and multi-app workflows.

5.2 Related Work

Multimodal Mobile Agents. Recent advances in large foundation models and the availability of extensive Android task datasets [179, 151, 121, 143] have driven significant progress in mobile agents that interact with real-world interfaces [166]. While these agents excel on in-distribution tasks, their generalization to unseen applications remains limited [40], highlighting a persistent challenge in GUI-based agent research [105, 22]. Studies show that the

trained agents often overfit to specific layouts and workflows, leading to poor performance on unseen or structurally different interfaces [22, 40]. Increasing model size or dataset diversity can further improve results on seen apps but has little effect on cross-application generalization [143]. Motivated by these limitations, we focus on improving generalization by introducing explicit procedural knowledge and training the agent to follow such procedures when its internal knowledge is insufficient.

Exploration and External Search for Agents. To mitigate the performance limitations arising from insufficient task knowledge in large foundation models, a growing body of work augments agents with exploration mechanisms [142, 69] or external information sources [179, 93]. When combined with large language models, external search improves task success on certain complex or long-horizon tasks. However, most existing methods invoke exploration or search unconditionally, either as a pre-execution phase [3] or as a fixed component of the agent pipeline [179]. In contrast, we focus on enabling agents to selectively seek external information through explicit meta-reasoning, allowing them to balance internal knowledge with external guidance based on task familiarity.

Hierarchical and Procedure-Guided LLM Agents. Hierarchical decomposition is a widely adopted strategy for long-horizon decision-making, where complex goals are broken down into intermediate subgoals or plans to improve reliability and reduce compounding errors [14, 99]. Recent LLM-based agents extend this paradigm by generating high-level plans that guide stepwise execution in interactive environments [163, 113, 141, 136], including GUI and web settings [26, 83]. While these approaches provide useful global structure, they primarily treat plans as execution scaffolds and focus on refinement or replanning in response to execution failures. In contrast, our work emphasizes procedural sufficiency as a distinct capability for agent control. Rather than using step-by-step knowledge solely for action decomposition, our agent maintains a complete task-level procedure and explicitly evaluates whether the remaining steps are plausible under the current UI context. This enables anticipatory identification of procedural deficiencies and selective triggering of external knowledge acquisition, a capability that is largely absent from prior hierarchical or procedure-guided agent architectures.

5.3 Method

5.3.1 Background

We study multimodal agents operating in Android environments as a representative and challenging experimental setting for real-world interaction. A foundation-model-based agent is tasked with completing a natural-language goal g by interacting with the Android user interface.

At each timestep t, the agent observes the current UI state as a screenshot o_t and executes an action a_t (e.g., tap, swipe, or text input), conditioned on the goal and the interaction history. Such an agent can be modeled as

$$a_t = \text{LMM}(g, o_t, a_1, \ldots, a_{t-1}), \tag{5.1}$$

where LMM is typically fine-tuned on expert demonstrations using supervised fine-tuning (SFT) [121, 143]. While these agents achieve strong performance on tasks and apps seen during training, their generalization to novel applications and distribution shifts remains limited. In this chapter, we aim to improve generalization under this standard training and evaluation paradigm.

5.3.2 Procedure-Aware Multimodal Agent with Meta-Reasoning

We propose **P**rocedure-**A**ware **M**ultimodal **A**gent with Meta-**R**easoning (PAMAR), a multimodal agent framework that improves generalization by explicitly decoupling procedural knowledge reasoning from grounded action execution. As illustrated in Figure 5.1, PAMAR consists of three components: (i) a high-level procedural generator M_p, (ii) a meta-reasoning module M_r that evaluates procedural sufficiency and governs external knowledge acquisition, and (iii) a low-level grounded actor M_a that translates abstract procedures into concrete UI actions, each instantiated as a multimodal foundation model.

During execution, the procedural generator M_p produces a task-level procedure, updating it online as the interaction unfolds. The meta-reasoning module M_r evaluates procedural sufficiency and determines when external search is required, providing additional information to refine the procedure when triggered. The grounded executor M_a conditions on the current procedure to execute UI actions sequentially.

5.3.3 Procedure-Aware Grounded Executor

Let p denote a task-specific procedure: an ordered list of natural-language steps specifying how to accomplish the goal. This procedure provides an explicit, human-interpretable representation of the agent's task understanding and supplies global structure for grounded execution. The low-level actor model M_a interacts directly with the environment by producing concrete UI actions given p.

5.3.3.1 Training with Procedural Supervision

We train the grounded low-level executor M_a with supervised fine-tuning (SFT) on procedure-augmented data:

$$a_t = M_a(g, o_t, a_1, \ldots, a_{t-1}, p), \tag{5.2}$$

so that it can reliably follow procedures when available. Conditioning on the full procedure throughout execution provides a global task structure that helps the executor ground individual actions in context.

5.3.3.2 Procedure Induction and Language Augmentation

To construct procedural supervision, we synthesize compact, temporally abstract natural-language procedures from expert demonstrations. Details are shown in Appendix D.2.2.

Transition-grounded step generation. Given a demonstration trajectory $\{(o_{t-1}, a_t, o_t)\}_{t=1}^{T}$, we first produce a fine-grained description for each atomic UI action. For each timestep, we prompt an LLM with the pre-action screenshot o_{t-1}, the executed action a_t, and the post-action screenshot o_t, and extract a concise text description d_t of the action intent grounded in the observed UI transition.

Step merging and normalization. The per-timestep descriptions d_t are often redundant due to retries and micro-actions. We therefore canonicalize surface form (e.g., normalizing tense and removing boilerplate) and merge consecutive descriptions that reflect the same short-horizon intent. For example, we merge repeated swipes and collapse retry taps by identifying sequences with the same screenshots where the tap coordinates are within a small pixel radius.

Condition-guided abstraction. To better match how humans express procedures and to reduce sensitivity to action counts, we further rewrite action-centric instructions into intent- and condition-guided forms when applicable. For example, we replace fixed-count scroll directives (e.g., *swipe up twice*) with state-dependent stopping conditions (e.g., *swipe a few times until the desired element appears*).

Procedure language augmentation. Finally, to improve robustness to linguistic variation at test time, we generate paraphrases for each procedure by prompting an LLM to rewrite the step descriptions we got from previous steps in alternative natural-language forms while preserving semantics. During training, we sample from the original induced procedure and its paraphrases when conditioning the grounded executor, exposing it to heterogeneous procedure styles (e.g., terse numbered instructions vs. narrative guidance).

5.3.4 High-Level Procedural Reasoner

The high-level procedural reasoner is responsible for producing and maintaining task-level procedural knowledge during execution. It consists of two components: a procedural generator and a meta-reasoning module.

5.3.4.1 Procedural Generator

At test time, the procedural generator M_p produces an initial natural-language procedure conditioned on the goal g and the initial observation. It then dynamically updates the procedure online at each timestep based on the current observation and interaction history, allowing it to adapt to interface changes and correct potential mistakes. Prompts are provided in Appendix D.2.3. Crucially, the agent is prompted to maintain a complete procedure throughout execution, enabling task-level sufficiency judgments about whether the remaining steps are plausible under the current context.

5.3.4.2 Procedure Meta-Reasoning

At each timestep t, the meta-reasoning module M_r assesses whether the procedure from the last step p_{t-1} remains valid and sufficient to achieve the task goal g given the interaction history $\{o_1, a_1, \ldots, o_{t-1}, a_{t-1}\}$, and outputs whether the agent can reliably proceed with it.

When M_r determines that p_{t-1} is not a good procedure and is not good enough to recover easily, it triggers an external knowledge acquisition tool call (e.g., web search). The retrieved information is then passed back to the procedural generator M_p to update the

procedure. This selective mechanism enables anticipatory knowledge augmentation while avoiding unnecessary tool use when the internally maintained procedure remains adequate.

We study two approaches for implementing this decision process: a prompt-based method that uses human-designed heuristics, and a training-based method.

Prompt-Based Meta-Reasoning. We implement M_r using a structured reasoning prompt that guides stepwise analysis over the ongoing episode with human heuristics. The prompt instructs M_r to assess progress with respect to p_{t-1}, verify whether the outcome of recent actions is consistent with expected UI changes, and check whether required UI elements for upcoming steps are present or reachable. Based on this assessment, M_r outputs a decision to either continue with the current knowledge or trigger external acquisition. Prompts can be found in Appendix D.2.4.

Training-Based Meta-Reasoner. In addition to prompting, we train M_r to predict when external acquisition is required. During training, we consider a case in which the external search provides perfect information for M_p to generate the ground-truth procedure. Under this assumption, we use the induced expert procedure $p^\star$ from Section 5.3.3.2 as the reference signal. We then prompt M_p to iteratively and dynamically generate a per-timestep procedure $\hat{p}_t$ under the same interaction context. This process yields an internal procedure that reflects the intrinsic task knowledge encoded in M_p as well as the information inferred from observations.

We derive supervision signals by comparing how well M_a explains expert actions under the two procedures. Specifically, let $\ell_t(p)$ denote the log-probability of generating the ground-truth action $a_t^\star$ under procedure p:

$$\ell_t(p) = \log P_{M_a}(a_t^\star \mid g, o_t, a_{<t}, p)\,. \tag{5.3}$$

We then define the margin $\Delta_t = \ell_t(p^\star) - \ell_t(\hat{p}_t)$. A large positive margin indicates that the reference procedure explains the expert behavior substantially better than the internally generated procedure, suggesting a procedural deficiency that motivates triggering external acquisition. Concretely, we label a timestep as requiring search when the action induced under $\hat{p}_t$ is incorrect, the action induced under $p^\star$ is correct, and Δ_t exceeds a fixed margin threshold.

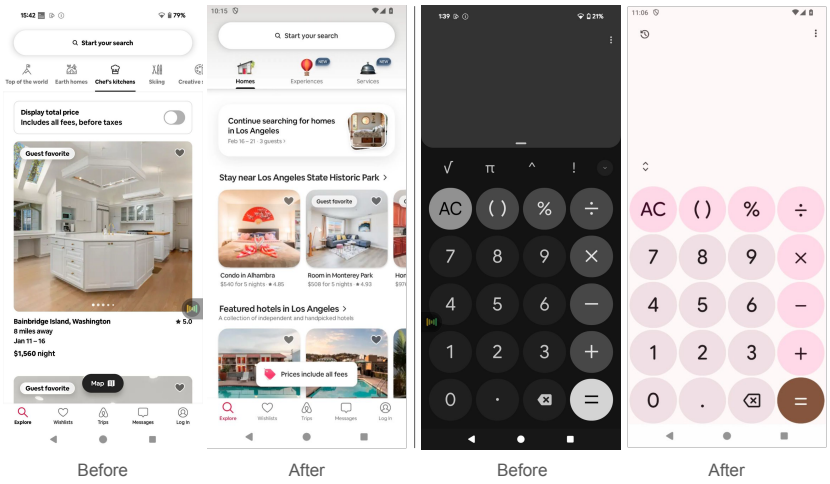


Figure 5.2: Examples of application version updates in DigiData-AppUpdate. Left: An Airbnb update removes category shortcuts below the search bar (e.g., *Top of the world*), breaking procedures that rely on these entry points. Right: A calculator redesign changes layout and visual structure while preserving core functionality, illustrating workflow/UI drift.

5.4 Experiments

5.4.1 Experimental Setup

Datasets. We evaluate PAMAR along three complementary dimensions of generalization.

First, we evaluate the agents on **DigiData-Bench-Auto** [143], which assesses both cross-task generalization on seen applications and cross-application generalization on previously unseen apps.

Second, we introduce **DigiData-AppUpdate**, which evaluates robustness to application version changes, including deprecated functionalities and UI/workflow shifts relative to the training data.

Third, we introduce **DigiData-MultiApp**. Agents are trained on single-app objectives, while evaluation episodes require completing tasks across multiple apps, stressing long-horizon planning and app transitions.

Baselines. Our primary baseline is an 8B-parameter Perception–Language Model, **PLM-8B** [19], fine-tuned for Android control. We follow the supervised fine-tuning (SFT) recipe of DigiData [143] and train on a mixture of four datasets: DigiData [143], AitW [121], AndroidControl [78], and Cauldron [75].

At test time, the policy predicts the next action given the task goal g, the current screenshot observation o_t, and the action history $a_{1:t-1}$: $a_t = \text{PLM}(g, o_t, a_{1:t-1})$. The action space

Method	Agent model	Cross-Task	Cross-App	All	Search(%)
Direct (no procedure)	PLM-8B	46.93	31.39	41.47	N/A
Direct (no procedure)	GPT-4o	11.80	3.93	9.00	N/A
PAMAR-Static	PLM-8B	49.83	39.88	46.30	N/A
PAMAR-Dynamic	PLM-8B	58.47	40.51	52.10	N/A
PAMAR-MR-Dynamic (Prompt)	PLM-8B	58.10	47.06	54.20	45.14
PAMAR-MR-Dynamic (Train)	PLM-8B	56.99	47.06	53.47	79.86
Always-search	PLM-8B	62.40	41.17	54.90	100.0

Table 5.1: Results on DigiData-Bench-Auto. The reported numbers are success rate (%) on cross-task, cross-app generalization tasks, and the whole evaluation set. Search(%) denotes the fraction of episodes that invoke external knowledge acquisition.

(e.g., tap, type, swipe) is defined in Appendix D.1.1.

In addition, we evaluate GPT-4o as a direct action generator.

Procedure-Augmented Agents. We train PLM-8B as the low-level grounded actor M_a using procedure-augmented trajectories, as described in Section 5.3.3. To isolate the effects of (i) conditioning execution on explicit procedures, (ii) enabling online procedure revision, and (iii) selectively acquiring external procedural knowledge, we evaluate three agent variants. All variants use GPT-4o as the procedural generator.

- **PAMAR-Static.** The agent generates a single procedure based on task goal and initial observation at the beginning of the episode using only internal knowledge, and follows this fixed procedure throughout execution.
- **PAMAR-Dynamic.** The agent may revise the procedure online using only internal knowledge, conditioning on the current observation and interaction history to correct UI references and adapt the workflow during execution.
- **PAMAR-MR-Dynamic.** In addition to online procedure revision, a meta-reasoner continuously assesses whether the current procedure is sufficient for task completion. External information acquisition (e.g., web search) is triggered only when a procedural deficiency is detected, after which the procedural generator produces an updated procedure that incorporates the retrieved information.

For the prompt-based meta-reasoner, we use GPT-4o with a structured reasoning prompt. For the training-based meta-reasoner, we fine-tune a Llama-4-Scout-17B-16E model using the supervision described in Section 5.3.4.2. Additional implementation details are provided

in Appendix D.2.4.

App documentation gathering. External acquisition is performed via app-specific knowledge search. Given the task goal g, we prompt an LLM to generate search queries targeting app-specific how-to instructions. When available, we query the official help center of the target app; otherwise, we fall back to FAQ-style support pages. Retrieved webpages are captured as screenshots to preserve multimodal information. The retrieved content is then processed by a knowledge accumulation module, which converts visual signals into textual descriptions and extracts task-relevant procedural snippets with an overlapping sliding-window scheme. In practice, the retrieved information is not assumed to be perfectly accurate and may be incomplete, partially relevant, outdated, or even misleading.

5.4.2 Cross-Task and Cross-Application Generalization

Setup. We evaluate cross-task and cross-application generalization using DigiData-Bench-Auto [143], which tests whether agents can transfer procedural knowledge to task goals and application contexts that differ from those observed during training. Due to the nature of this benchmark, where tasks have clearly specified goals and relatively short horizons (at most 25 interaction steps), we restrict agents to searching at most once per episode. When search is triggered, the retrieved information remains available for all subsequent steps in the trajectory.

Results. Table 5.1 reports success rate and the fraction of episodes that invoke search. Conditioning the grounded actor on explicit task procedures improves generalization, with consistent gains in both cross-task and cross-application settings. Notably, even a static procedure generated once at the start helps, suggesting that explicit procedural structure provides global guidance beyond memorized action sequences; updating procedures online with internal knowledge (PAMAR-Dynamic) yields additional gains, especially for cross-task generalization, indicating benefits from adapting task-level structure during execution.

External information further strengthens generalization: both always-search and MR agents outperform procedure-only variants overall, implying that retrieval supplies useful task- and application-specific knowledge in unfamiliar tasks and unseen applications, beyond what the procedural generator alone provides. At the same time, selective acquisition is substantially more efficient. Compared to unconditional retrieval, PAMAR-MR-Dynamic reduces search usage from 100% to 45.14% of tasks, while maintaining comparable success.

Finally, the training-based meta-reasoner exhibits more conservative search behavior than the prompt-based variant, triggering acquisition more frequently. This is expected because

training assumes retrieved information is high-quality and consistently helps recover expert-induced procedures, biasing the policy toward searching under procedural ambiguity. In addition, training-time supervision compares action likelihoods to a single reference trajectory in a static dataset; since multiple valid action sequences may solve the same goal, reasonable deviations can be penalized, further encouraging search. Despite this tendency, the results show that procedural sufficiency signals can be learned from data.

5.4.3 Application Version Update Adaptation

Setup. Mobile applications are frequently updated, often introducing changes that invalidate previously correct task procedures. This is also true for applications in DigiData-Bench-Auto [143]. Since the original DigiData demonstrations were collected, many target apps on the emulators have undergone version updates that break or alter existing workflows. To isolate this source of distribution shift, we construct **DigiData-AppUpdate**. More specifically, we identify two prevalent update patterns: functionality deprecation, where a previously supported feature is removed, and workflow or UI drift, where the task remains supported but its interface or navigation structure changes. Examples are shown in Figure 5.2. From the DigiData training pool, we select 61 training task goals on seen apps whose original completion trajectories are affected by such updates and evaluate them on updated app versions.

In this setting, external information corresponds to update-specific information, such as migration hints or release notes. Because reliable public documentation is unavailable for most tasks, we provide curated human-written update notes as the external retrieval results for this experiment.

Results. Table 5.2 reports success rates and search frequencies on DigiData-AppUpdate. The action-only baseline performs poorly, confirming that post-training application updates can invalidate previously learned execution patterns even on seen apps. Adding a static procedure provides little benefit, as the LLM-based procedure generator itself may also encode outdated procedural knowledge.

Allowing dynamic procedure revision substantially improves success, indicating that conditioning high-level guidance on the current UI state can partially compensate for interface drift. Incorporating external update notes yields the highest performance: PAMAR-MR-Dynamic recovers from cases where critical task knowledge is missing due to deprecated features or altered workflows, while avoiding unnecessary retrieval by triggering external acquisition only when procedural insufficiency is detected.

Method	SR(%)	Search(%)
No procedure	29.50	N/A
PAMAR-Static	30.35	N/A
PAMAR-Dynamic	55.75	N/A
PAMAR-MR-Dynamic (Prompt)	59.03	39.30
PAMAR-MR-Dynamic (Train)	63.93	78.69

Table 5.2: Results on DigiData-AppUpdate. SR(%) denotes success rate. Search(%) is the fraction of episodes that query external update notes. Procedures are generated with GPT-4o.

The training-based meta-reasoner retrieves external knowledge more frequently than the prompt-based variant, but also achieves higher success. This behavior is expected, as our training objective assumes that retrieved information reliably enables recovery of the expert procedure, biasing the learned policy toward conservative retrieval. In this setting, where external knowledge is human-curated, the apps are seen during training, and many interface structures and UI elements remain unchanged after the update, such behavior leads to improved task completion.

5.4.4 Compositional Multi-App Generalization

Setup. Real-world mobile objectives frequently require coordinating information and actions across multiple applications (e.g., retrieving information from system settings and acting on it in a browser). However, most Android agent training data consists of single-app trajectories, encouraging agents to overfit to within-app workflows and making app-to-app transitions particularly challenging.

To evaluate this setting, we construct **DigiData-MultiApp**, a focused benchmark of 20 compositional tasks that require completing a goal across two applications. Each episode requires extracting task-relevant information from a source app and subsequently using a target app to complete a dependent action. For example, *in Phone by Google, find which carrier is currently used for outgoing calls; then open Google Chrome and search for deals for that carrier.*

Results. Results are shown in Table 5.3. Single-step action policies fail completely in this setting, indicating that agents trained solely on single-app trajectories struggle to initiate app transitions or preserve intermediate information across applications. Adding a static procedure does not improve performance, as a fixed plan is insufficient for long-horizon tasks

Method	SR(%)	Search(%)
PLM-8B (No procedure)	0.0	N/A
PAMAR-Static	0.0	N/A
PAMAR-Dynamic	10.0	N/A
PAMAR-MR-Dynamic	15.0	60.0

Table 5.3: DigiData-MultiApp Results. These tasks require navigation across two applications. SR(%) denotes the percentage of successfully completed episodes, while Search(%) denotes the percentage of episodes that require querying external information.

that require long-term memory. Allowing dynamic procedure revision enables some success, demonstrating that adapting high-level guidance during execution can partially support cross-app reasoning. Although absolute performance remains low due to the difficulty of the benchmark, PAMAR-MR-Dynamic achieves the highest success rate, suggesting that explicit procedural guidance and selective retrieval can be beneficial even in compositional multi-app settings. Notably, the meta-reasoner continues to invoke retrieval selectively rather than defaulting to retrieval in every episode, indicating that retrieval decisions remain controlled as task complexity increases.

5.4.5 Ablation

We perform ablation studies to analyze the key components and training choices of PAMAR.

5.4.5.1 Training–Test Procedure Alignment

We observe that grounded procedure following should be learned rather than imposed purely by prompting at test time. In particular, if the executor is trained only on action-only demonstrations, providing a procedure at test time may not help and can even be harmful, as the model is not calibrated to interpret procedural language. Moreover, even when training includes procedures, robustness may depend on the diversity of procedural language seen during training.

To test these hypotheses, we vary the form of procedural supervision during training and evaluate with and without procedures at test time. Concretely, we compare: (i) action-only training, (ii) training with step-wise procedures generated from demonstrations with simple step merging and normalization, and (iii) training with step-wise procedures plus our full procedure abstraction and augmentation. At evaluation time, we measure success on novel applications under two conditions: without procedures (standard actor setting) and with procedures (procedure-conditioned execution).

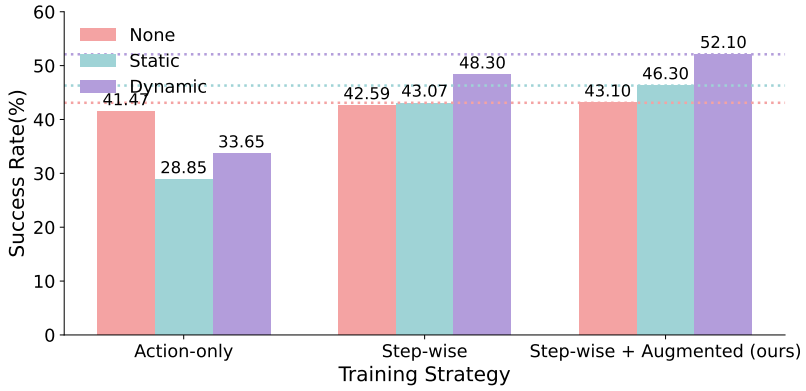


Figure 5.3: **Ablation Study: Training–Test Procedure Alignment**. Success rates (%) on DigiData-Bench-Auto for three evaluation settings: no procedure (None), static procedure (Static) and dynamic procedure (Dynamic). All procedures are generated with GPT-4o.

Figure 5.3 highlights two findings. First, when the executor is trained without procedural supervision, adding procedures at test time does not reliably help and can hurt performance, consistent with a train–test mismatch in inputs. Second, step-wise procedural supervision improves procedure-conditioned execution, but a large gap to the fully augmented procedure set remains, suggesting sensitivity to procedure phrasing/format. Further augmentation yields the biggest gains, indicating that exposure to diverse procedural variants is important for robust test-time following.

5.4.5.2 Meta-Reasoner Trigger Timing

Our framework relies on the meta-reasoner to decide *when* external procedural knowledge should be acquired. To test whether the *timing* of these triggers matters, we perform a controlled test-time intervention that delays retrieval.

Intervention. For the training-based meta-reasoner, when the model predicts that external acquisition should be triggered, we suppress the *first* such trigger in an episode with 50% probability, forcing the agent to continue execution using its internally maintained procedure. To avoid trivially removing retrieval for the remainder of the episode, we add a fallback: if no retrieval is executed for the next 5 decision steps, we force a retrieval call at the next step. All other components, including the procedural generator, actor, and the retrieval mechanism itself, are kept identical.

Results. The delayed-trigger intervention reduces success rate on DigiData-Bench-Auto from 53.47% to 48.61% (Table 5.4). The performance drop indicates that when retrieval is performed matters: delaying acquisition can cause the agent to enter incorrect UI states or

Method	SR(%)
PAMAR (Train)	53.47
PAMAR (Train, Delayed Trigger)	48.61

Table 5.4: **Ablation: Meta-Reasoner Trigger Timing.** Success rate on DigiData-Bench-Auto (All split). Delaying the first predicted retrieval trigger noticeably degrades performance.

take hard-to-reverse actions, after which later retrieval becomes less effective. Notably, this result is complementary to the strong performance of the always-search variant reported in Table 5.1, which shows that external knowledge can be highly beneficial. However, always-search indiscriminately performs retrieval even when the internally maintained procedure is already sufficient, which can lead to unnecessary tool usage and latency in interactive settings. Taken together, these results suggest that effective deployment of retrieval requires not only access to external knowledge, but also appropriate trigger timing.

5.5 Conclusion

In this chapter, we presented PAMAR, a procedure-aware multimodal agent that improves generalization by explicitly reasoning about the sufficiency of task-level procedural knowledge during execution. By decoupling high-level task knowledge from low-level grounded action generation and selectively acquiring external information only when internal procedures are insufficient, our approach avoids the brittleness of unconditional retrieval. Experiments on challenging Android navigation tasks demonstrate substantial generalization gains across unseen tasks and applications, interface changes, and multi-app workflows, while reducing unnecessary external tool usage. These results highlight procedural sufficiency as a useful abstraction for building robust interactive agents.

CHAPTER 6

Against Epistemic Shortcuts: Evidence-Grounded Verification for LLM Agents

6.1 Introduction

Modern LLM-based agents are becoming increasingly powerful with the advances of large language models [170, 132, 184], and are being applied to diverse domains [188, 56]. Their performance is commonly evaluated by task completion: whether the agent eventually reaches the target state or answers the question correctly. While outcome-based evaluation is convenient and often necessary, whether the corresponding trajectory is self-contained and reliable remains uncertain [131, 62]. Consider the following cases. If an agent is asked to answer how many comments on a video have more than 10 likes, it may not inspect all pages, but all qualifying comments may happen to appear on the first page; in this case, an answer of *5* would still be judged successful. If an agent is asked to buy a gluten-free bagel, it may not actually verify whether the selected item is gluten-free, but instead make a rough selection based on its prior or internal knowledge and still succeed. These cases show that a trajectory can complete the task without every action being properly justified by the information available to the agent. In other words, an agent can succeed without knowing enough to act.

In this chapter, we refer to such behavior as an **epistemic shortcut**: an action that depends on assumptions that have not been verified with the observed interaction history. Such behavior is not only a common cause of failed trajectories but can also appear in *successful* trajectories that complete the task. In the former case, agents may hallucinate and make unverified assumptions, leading to incorrect conclusions and limiting their ultimate success. The latter case can be even more problematic, as it makes it difficult to measure agent performance faithfully. Moreover, if such collected trajectories are used for further purposes, such as training data or in-context learning examples, these unsupported or ambiguous action patterns, which lack sufficient reasoning support, may be passed on to agents at test time. This is a common data-collection problem, and it is not limited to agent-collected tra-

jectories. Human demonstrations can also contain epistemic shortcuts, since humans often rely on environment-specific priors, multimodal cues, or memories that are not recorded in text-based trajectories.

Existing approaches to improving LLM agents often focus on better planning, self-reflection, or outcome-level optimization. These methods can improve task success, but they do not ensure that the trajectory is fully explainable and faithful. Thus, in this chapter, we propose EVIA, Evidence-Grounded Verification for Interactive LLM Agents, an evidence-driven framework for detecting and correcting epistemic shortcuts in LLM agents. The core component of EVIA is an **Epistemic Monitor** that intercepts a proposed shortcut action. The monitor reasons about the action's factual prerequisites, which are the facts that must hold for the action to be justified, and verifies whether each prerequisite is explicitly supported by the observations available in the trajectory. Using this monitor, EVIA distinguishes *verified successes* from *pseudo-successes*: trajectories that complete the task while relying on unsupported critical actions. For pseudo-successful trajectories in the training data, we use the monitor's feedback to propose alternative actions and re-gather the subsequent trajectories, so that success is achieved through verified evidence rather than unsupported assumptions. This process converts successful-but-unverified trajectories into evidence-grounded demonstrations, making them more suitable for agent learning. Experiments on LOCA-bench [176], WebShop [169], and PackAudit, a controlled package-approval environment that we developed, demonstrate the effectiveness of EVIA across different settings.

In summary, our contributions are as follows:

- We identify epistemic shortcuts as a common failure mode in LLM-agent trajectories, in which actions rely on assumptions that are not grounded in the observed interaction history.
- We propose EVIA, with an Epistemic Monitor that evaluates action-level evidence sufficiency by decomposing proposed actions into factual prerequisites and verifying whether they are supported by the trajectory. EVIA can be used for both test-time control and training-time rectification.
- We empirically demonstrate that EVIA improves agent reliability and final task performance across several domains with varying levels of difficulty and diversity.

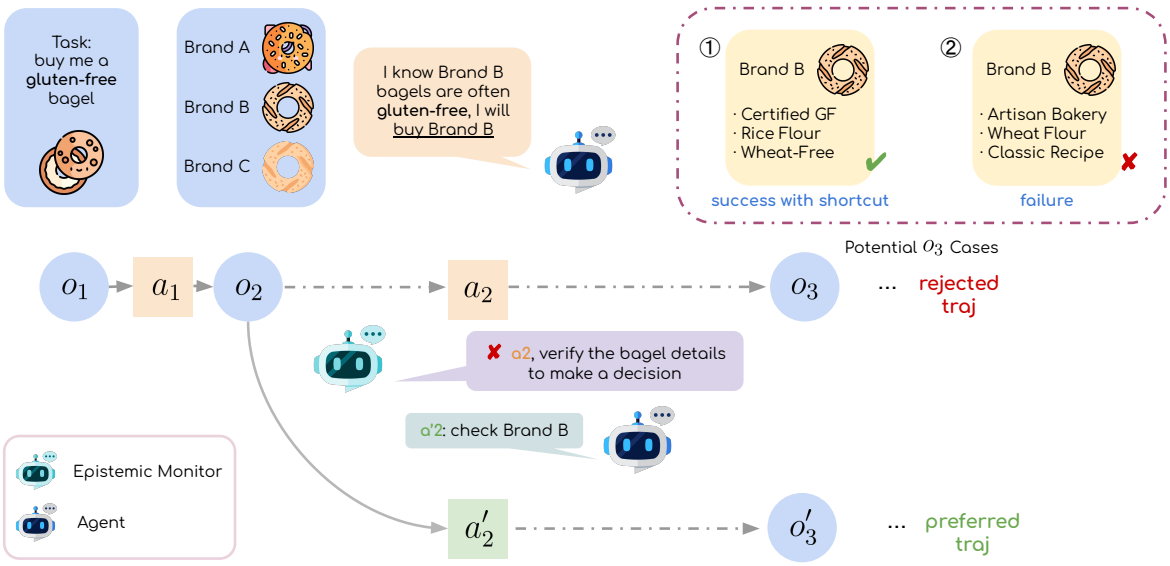


Figure 6.1: Illustration of how EVIA works. In this example, the agent hallucinates that Brand B is gluten-free based on its internal knowledge, without specifically checking whether this bagel is gluten-free. If it is, this leads to a successful trajectory with a shortcut; otherwise, it directly causes the trajectory to fail. The epistemic monitor detects that the action *Buy Brand B* is not supported by sufficient environmental evidence, rejects the action, and asks the agent to regenerate. This either works directly at test time to prevent shortcut behavior, or naturally constructs preference pairs for further fine-tuning.

6.2 Related Work

Self-correction and verification in LLMs. Self-reflection and critique have been widely explored as means of improving LLM performance. Existing methods prompt LLMs to generate trajectory-level or output-level reflections that inform subsequent attempts or iterative refinement [132, 122, 92]. More recently, evidence-grounded verification of reasoning and chain-of-thought in question-answering tasks has received increasing attention. For example, SAFE [72] uses a trained verifier to determine whether atomic reasoning steps are supported by the available evidence. SAVER [173] proposes a test-time framework that generates diverse candidate belief states and audits their logical and evidential support before final answer generation. In contrast, the Epistemic Monitor in EVIA focuses on actions in interactive environments. It verifies whether a proposed action is sufficiently supported by the observations and redirects the agent toward further information gathering when necessary. This action-level verification also helps produce evidence-grounded and interpretable trajectories that expose more comprehensive environment signals for agent learning and generalization.

Epistemic failures in LLM-based agents. Recent studies have documented a range of epistemic limitations in LLM agents [62, 160, 131]. Agents may fail to recognize that the available evidence is insufficient and prematurely commit to an action [62, 131], which can arise from LLM hallucinations [183, 87]. Epistemic reliability may also deteriorate over long-horizon interactions. In particular, growing contexts can substantially degrade agent performance and make it difficult to adhere to plans established earlier in the trajectory [176, 182]. Moreover, even when insufficient evidence is recognized or exposed, agents may still struggle to recover by identifying effective information-seeking actions [131, 35]. EVIA addresses these complementary failure modes by detecting unsupported actions, redirecting agents toward evidence-seeking behavior, and using the resulting supervision to improve agent training.

Agent training with improved data quality. Whether through parameter updates [103, 53, 119] or the extraction of natural-language knowledge [184, 82], improving agents requires high-quality data. Approaches to improving trajectory quality span multiple perspectives, from carefully filtered demonstrations [174] to outcome-based [137] preference data or preference pairs constructed around verified critical steps [77], and from injecting noise or perturbations into agent trajectories [160, 34] to replacing error-contaminated segments with successful corrections [164]. EVIA identifies data problems from another perspective: epistemic shortcuts. We introduce the notion of shortcut success to distinguish it from genuine success and construct preference data accordingly.

6.3 Method

6.3.1 Problem Setting

We consider an interactive LLM-based agent that operates within an environment through a sequence of actions to complete a task g. At each timestep t, the agent receives an observation o_t from the environment and maintains an interaction trajectory: $\tau_t = (o_1, a_1, o_2, a_2, \ldots, o_t)$, where a_i denotes the action executed at timestep i. Given the current trajectory τ_t, the base agent π_θ proposes the next action: $a_t \sim \pi_\theta(\cdot \mid g, \tau_t)$ to interact with the environment.

In standard agent training and evaluation paradigms, a trajectory is typically deemed successful if it achieves the desired final outcome. However, an action may rely on assumptions that remain explicitly unverified within the observed interaction history. We designate such actions as **epistemic shortcuts**. Formally, an action a_t may require a set of factual prerequisites:

$$\mathcal{P}(g, a_t, \tau_t) = \{p_1, p_2, \ldots, p_m\}, \tag{6.1}$$

where each prerequisite p_i represents a fact that must hold for the action to be rigorously justified. The action is considered evidence-supported if every prerequisite is fully supported by evidence in the trajectory. Otherwise, if any prerequisite p_i remains unverified, an epistemic shortcut occurs. Such an action may precipitate immediate failure, or it may serendipitously lead to task success, resulting in an imperfect trajectory demonstration that contains shortcuts and potentially encourages misleading or guess-based action patterns.

6.3.2 Evidence-Grounded Verification for Interactive LLM Agents

We propose EVIA—**E**vidence-Grounded **V**erification for **I**nteractive LLM **A**gents, an evidence-driven framework designed to detect and correct epistemic shortcuts within LLM-agent trajectories. Rather than evaluating an action solely by final success, EVIA verifies whether the agent possesses sufficient contextual evidence to justify executing that action.

6.3.2.1 Epistemic Monitor

The core component of EVIA is an *Epistemic Monitor*. Given the task instruction g, the current trajectory τ_t, and a proposed action a_t, the monitor yields three distinct outputs:

$$M(g, \tau_t, a_t) = (v_t, \mathcal{U}_t, f_t), \tag{6.2}$$

where $v_t \in \{\textsc{Supported}, \textsc{Unsupported}\}$ denotes a verification verdict, $\mathcal{U}_t$ represents the set of unverified prerequisites, and f_t constitutes natural-language feedback elucidating the missing evidence. If $v_t = \textsc{Supported}$, the proposed action is permitted to proceed. Conversely, if $v_t = \textsc{Unsupported}$, the action is classified as an epistemic shortcut and is subsequently rejected with f_t as reasoning behind.

Specifically, the monitor executes the following reasoning procedure:

1. **Prerequisite identification.** Given (g, τ_t, a_t), identify the set of facts $\mathcal{P}_t = \{p_1, \ldots, p_m\}$ that must hold true for a_t to be justified.

2. **Evidence grounding.** For each prerequisite p_i, determine whether it is directly observed or validly entailed by the observations in τ_t.

3. **Shortcut detection.** If any prerequisite remains unsupported, output $\textsc{Unsupported}$ and enumerate the unverified prerequisites $\mathcal{U}_t \subseteq \mathcal{P}_t$.

4. **Feedback generation.** Formulate feedback f_t that articulates why the action lacks current justification and specifies the information the agent must subsequently acquire.

This architectural design differs from approaches that guide action selection using a progress-based reward function. Such reward functions may assign positive credit to an action even when the available environmental observations are insufficient to justify that action. Consequently, the agent may learn to generate unsupported or guess-based actions, potentially introducing misleading action patterns and degrading test-time performance.

6.3.2.2 Test-Time Intervention

The most straightforward application of the epistemic monitor is its integration as an auxiliary module during test time. At each step, the base agent initially proposes an action a_t. Prior to execution, the epistemic monitor evaluates whether the proposed action is sufficiently supported by the trajectory. If the monitor returns SUPPORTED, the action is executed nominally. Otherwise, the proposed action is rejected. The monitor's feedback, f_t, is then appended to the agent's context, serving as supplementary reasoning to guide the new action generation $a'_t \sim \pi_\theta(\cdot \mid g, \tau_t, f_t)$. This feedback mechanism encourages the agent to either acquire the missing evidence or elect an alternative action that is already supported by the available observations.

6.3.2.3 Data Rectification and Preference Learning

While test-time intervention effectively prevents shortcut actions dynamically, calling the epistemic monitor at every step increases token consumption and inference latency. To eliminate runtime overhead, we utilize the monitor offline to audit collected trajectories, amend shortcut behaviors, and construct preference pairs for policy fine-tuning. This training pipeline operates in three simple steps:

1. Shortcut Identification. Given a task-level successful trajectory $\tau = (o_1, a_1, \ldots, a_{T-1}, o_T)$ for task goal g, we audit each action a_t using the epistemic monitor M. If all actions are evidence-supported ($v_t = \text{SUPPORTED}$), τ is classified as a *verified success*. If any action a_t relies on unverified prerequisites ($v_t = \text{UNSUPPORTED}$), τ is identified as a *pseudo-success*.

2. Feedback-Guided Rectification. For a pseudo-successful trajectory, we leverage the monitor's feedback f_t as explicit reasoning to rectify the shortcut action. Prompted with the trajectory prefix τ_t and feedback f_t, an LLM generates an amended, evidence-grounded action: $a'_t \sim \pi_{\text{LLM}}(\cdot \mid g, \tau_t, f_t)$. The agent then re-executes the environment from a'_t to gather a rectified trajectory τ^+.

3. Preference Pair Construction. Under the trajectory context (g, τ_t), the original shortcut action and the rectified action naturally form a preference pair, where the amended action $y_t^+ = a'_t$ serves as the chosen response and the original shortcut action $y_t^- = a_t$ serves as the rejected response. These preference pairs can be used to optimize the policy with any preference learning method, such as Direct Preference Optimization (DPO) [119]. Through preference learning, the policy learns to favor evidence-supported actions over shortcut actions, thereby internalizing evidence-grounded decision-making without incurring monitor overhead.

6.4 Evaluation

6.4.1 Experimental Setup

6.4.1.1 Baselines

We compare EVIA against the corresponding base agent without epistemic monitoring.

For the training-time experiments, we compare supervision derived from EVIA against the following baselines: **SFT** on demonstrations and **Step-DPO** [73], adapted to the agent setting using outcome-based preference pairs.

In all test-time experiments, the baseline agent is a ReAct agent. We additionally evaluate EVIA, which introduces an additional monitor module on top of **ReAct** [170]. Note that this additional module is not applied when evaluating agents trained with EVIA.

6.4.1.2 Benchmarks

We evaluate EVIA in three domains:

- **LOCA-bench.** We use LOCA-bench [176] for both training-time and test-time evaluation. LOCA-bench evaluates agents under controlled context growth and exposes insufficient exploration as a salient failure mode. In particular, as context accumulates, agents may terminate exploration prematurely and mistake partial evidence for a complete review. This behavior closely aligns with the epistemic shortcuts studied in this chapter, making it a suitable evaluation testbed.

- **PackAudit.** We construct a controlled package-approval environment to directly test the extent to which agents suffer from epistemic shortcuts. In each task, the agent must decide whether to approve or reject a software package by navigating through folders and files. Additional details are provided in Appendix E.

	LOCA-bench	
Configuration	**8K**	**16K**
ReAct	61.3	36.0
SFT	60.0	42.7
Step-DPO	61.3	57.3
EVIA-DPO	**73.3**	**65.3**

Table 6.1: Mean accuracy on LOCA-bench across different context-length settings for ReAct, SFT, Step-DPO, and EVIA-DPO.

- **WebShop.** WebShop [169] is a text-based shopping environment in which an agent must search for and purchase a product that satisfies a user instruction. Agents in this environment may select an item based on an apparent match in its title without more carefully verifying the required attributes or features.

6.4.1.3 Models

For the training-time experiments, we use Qwen3.5-4B as the monitor to collect data and train agents based on the same backbone LLM. For all test-time experiments, we use Qwen3.5-35B-A3B as both the base agent model and the monitor.

6.4.2 Training with Monitor-Derived Supervision

We first investigate whether the trajectory corrections produced by EVIA can provide useful training supervision. A standard outcome-based data-collection pipeline treats successful trajectories as positive examples. However, this procedure may retain pseudo-successful trajectories in which the final outcome is correct even though the agent did not verify the evidence required for one or more critical actions. In contrast, EVIA applies a training-time monitor to identify epistemic shortcuts, provide reasoning to guide action regeneration, refine the trajectories, and formulate preference pairs.

As shown in Table 6.1, EVIA-DPO consistently outperforms all other configurations across the evaluated context lengths, demonstrating the effectiveness of monitor-derived supervision even in this challenging long-context setting.

6.4.3 Test-Time Epistemic Monitoring

As our epistemic monitor can be easily incorporated as an additional test-time module, we next evaluate whether EVIA improves action selection when applied directly at inference

Domain	Setting	ReAct	EVIA
PackAudit	Outdated summary	30.0	76.0
	Updated summary	100.0	100.0
LOCA-bench	8K	73.3	80.0
	32K	44.0	53.3
	128K	14.7	18.7
WebShop	–	47.0	48.4

Table 6.2: Test-time success rates (%) of ReAct and EVIA on PackAudit and WebShop and mean accuracy on LOCA-bench.

time and examine the extent to which shortcut behaviors influence final outcomes.

Table 6.2 summarizes the test-time results across PackAudit, LOCA-bench, and WebShop.

On PackAudit, we consider two settings, updated summary and outdated summary, corresponding to whether the first page cached summary is consistent with the ground-truth conditions or not. For the updated-summary setting, both ReAct and EVIA perform well because even relying on the external summary without navigating through the files does not affect the outcome. However, when the summary does not match the actual contents, ReAct begins to suffer from shortcut behaviors, resulting in a lower final success rate. On both LOCA-bench and WebShop, we observe a similar trend of improvement, demonstrating that EVIA generalizes well to more challenging environments.

Overall, the results demonstrate two complementary benefits of epistemic monitoring. At training time, EVIA distinguishes evidence-grounded successes from pseudo-successes and converts this distinction into informative supervision that improves test-time performance. At test time, EVIA improves decision quality by preventing the agent from acting on incomplete or outdated evidence and reduces potential failures caused by epistemic shortcuts.

6.5 Conclusion

In this chapter, we study epistemic shortcuts in LLM-based agents: agents may act on assumptions that are not verified by the observed trajectory, leading either to successful final outcomes achieved through shortcuts or to task failures. To address this issue, we propose EVIA, an evidence-driven framework equipped with an Epistemic Monitor that checks whether the factual prerequisites of a proposed action are grounded in the available observations. EVIA can be used both to intervene at test time and to rectify pseudo-successful trajectories into evidence-grounded supervision at training time. Experiments on LOCA-bench, WebShop, and PackAudit show that EVIA helps distinguish verified success from

shortcut-based pseudo-success, improves evidence-grounded decision-making, and enhances overall agent performance.

CHAPTER 7

Conclusion and Future Work

7.1 Key Findings and Contributions

This dissertation studies interactive agents from the perspectives of effective agent learning and reliable adaptation to test-time environments. It begins by studying how agents can acquire and reuse task knowledge in natural language form. In Chapter 2, we introduce AutoGuide, a framework that automatically summarizes desired behaviors from offline trajectories into context-aware guidelines. At test time, AutoGuide retrieves guidelines based on the current context, enabling strong performance in challenging domains such as real-world web navigation. Despite this success, agents may struggle when the test-time environment shifts or evolves, or when the provided guidelines are incomplete, misleading, or ambiguous. In such cases, they may fail to identify the mismatch and recover appropriately. To better understand this limitation, we introduce MIRAGE in Chapter 3, a benchmark for evaluating the robustness of LLM agents under imperfect guidance. MIRAGE covers multiple interactive domains and four different types of guidance imperfections, revealing important failure modes of current LLM agents. It also highlights the importance of reasoning about when task knowledge should be trusted, ignored, or supplemented through interaction.

In Chapters 4–6, we consider different perspectives for improving agents' adaptation abilities. Chapter 4 shifts the focus from using prior knowledge to actively exploring the environment when such knowledge is insufficient. We first introduce GoBI, a sample-efficient reinforcement learning algorithm for procedurally generated, sparse-reward environments. GoBI uses a learned world model to imagine reachable states and provides intrinsic rewards for approaching states that expand episodic reachability. Agents trained with this objective learn to actively explore test-time environments with unseen layouts. These principles also offer a useful perspective for LLM-based agents, which are typically neither trained nor prompted to actively explore when their knowledge of the environment is insufficient. Through a simple Best-of-N mechanism that dynamically selects between task progression and exploration,

we provide preliminary evidence that active test-time exploration is a promising direction for LLM-based agents. Chapter 5 considers a more realistic and challenging setting, where we aim to develop general-purpose multimodal mobile agents. In real-world mobile environments, agents face diverse applications and frequent interface updates, which require strong adaptation abilities. We propose PAMAR, a procedure-aware multimodal agent that decomposes action generation into procedural planning and multimodal grounding. By meta-reasoning about whether the model's internal knowledge is sufficient to complete the task, PAMAR can selectively ask for external information to further improve performance. Finally, in Chapter 6, we introduce the notion of epistemic shortcuts, which are actions that lack sufficient environment support and may lead to direct failure or pseudo-successful trajectories. We discuss how to mitigate these shortcuts at test time through evidence-grounded verification, and how the epistemic monitor can be applied to amend demonstrations with shortcuts and improve the quality of training data.

Taken together, these works offer several important insights. First, both task knowledge and contextual information can be effectively represented in natural language, and the generation and application of context-aware guidelines can improve decision-making. Moreover, the way agents are trained is also critically important. By decoupling planning from grounding and carefully augmenting or refining training data, agents can develop a strong capacity to adapt to challenging test-time environments, including multimodal, real-world settings. In addition, enabling agents to determine when exploration is necessary and how to gather relevant information from their environments provides an effective means of improving test-time performance. By integrating these directions, this dissertation contributes to a broader vision of interactive agents that can learn and adapt reliably.

7.2 Future Work

Despite the advances presented in this dissertation, building fully reliable interactive agents remains an open challenge. Several promising directions naturally emerge from the work presented in this dissertation:

Toward agents that systematically learn and apply exploration behaviors. In Section 4.5, we already presented a simple experiment using Best-of-N with both progress and exploration mechanisms through LLM prompting. However, prediction of precise environment-specific value would be useful to further improve performance. Following GoBI [33], we can derive agent exploration rewards based on whether new task-specific beliefs can be imagined or derived from new state visitation, and train an exploration value prediction model

accordingly. This reward can also be directly applied to reinforcement learning algorithms that jointly optimize exploration and task returns to improve sample efficiency.

Learning from noisy and heterogeneous guidance. PAMAR [34] takes an initial step toward improving robustness to imperfect procedural knowledge by introducing LLM-based procedure rewriting as a data augmentation strategy. By exposing the grounded agent to procedures with different expressions and mild perturbations during training, the agent becomes less sensitive to the exact form of procedural knowledge provided at test time. Future work should move beyond robustness to surface-level procedure variation and study how agents can systematically reason about the usefulness and reliability of different test-time resources. One possible direction is to train agents to reason explicitly about what should be learned from noisy guidance. Such reasoning traces could either be proposed by LLMs and optimized using RL rewards, or automatically constructed from environment perturbations and used as supervision for fine-tuning.

Toward Lifelong Learning in Interactive Agents. The long-term goal of this line of work is to build agents that improve through continuous interaction. Such agents should accumulate task knowledge over time, organize it into reusable forms, retrieve relevant knowledge when needed, detect when existing knowledge becomes outdated, and revise their behavior based on new evidence. AutoGuide [32] provides a starting point by organizing and retrieving guidelines through context-based keys. However, as the number and diversity of tasks grow, these keys may become insufficiently discriminative, and faithfully updating guidelines through long-term interaction remains an open challenge. Future work could integrate the methods introduced in this dissertation into lifelong agent architectures that maintain and continually refine task knowledge. A central challenge is preventing lifelong agents from accumulating unreliable, contradictory, or outdated information. Addressing this challenge requires mechanisms for knowledge consolidation, conflict resolution, selective forgetting, and continual verification.

APPENDIX A

Automated Generation and Selection of Context-Aware Guidelines for Large Language Model Agents

A.1 Limitation and Broader Impacts

Limitation. The performance of AutoGuide depends on the diversity of offline experiences. As such, one important direction for improvement is to automatically collect diverse offline experiences through continual learning, where we iteratively generate guidelines and use them to gather more trajectories with high rewards. Another avenue is the need for quantifying the quality of generated contexts and guidelines. Currently, apart from applying context-aware guidelines to ReAct and measuring test time performance, there lacks a standardized method for quantifying the quality of generated contexts and selected guidelines. Introducing a quantifiable metric to approximate the quality could pave the way for new optimization approaches such as reinforcement learning.

Broader impact. This work introduces research aimed at enhancing the decision-making capabilities of LLMs. In terms of societal impact, while we develop a generic LLM-based autonomous agent, having biased offline datasets may lead to making decisions with suboptimal outcomes. Additionally, autonomous agents may be misused for malicious applications. To mitigate these risks, potential solutions would include diversifying datasets, implementing ethical oversight, ensuring transparency and accountability, engaging with stakeholders for a broader perspective, and incorporating security measures to prevent misuse. We believe that the research community, including ourselves, should responsibly advance LLM-based agent research, prioritizing societal well-being and ethical considerations.

A.2 Evaluation Details

A.2.1 ALFWorld [135]

A.2.1.1 Environment Details

Each task in ALFWorld starts with a description of the specific environment and the goal to achieve. At each timestep, an agent can choose one of the following actions to interact with the objects and receptacles in the environment:

- go to [recep]
- take [object] from [recep]
- put [object] in/on [recep]
- open/close/use [recep]
- clean/heat/cool [object] with [recep]

Alternatively, the agent can generate a think action for planning and reflection, which helps with decision-making but does not change the environment itself. After one action is performed, the environment returns an observation that describes view changes.

Following ReAct, we concatenate a list of (observation, action) pairs to show the entire trajectory up to the current timestep for LLM agents to generate the next action. We experiment on 134 unseen test tasks with 6 categories of `pick_and_place`, `pick_clean_then_place`, `pick_heat_then_place`, `pick_cool_then_place`, `look_at_obj`, and `pick_two_obj`. For each task, the agent is allowed to take a maximum of 50 actions.

A.2.1.2 Baseline and Models

For ALFWorld tasks, we follow the same setting as ReAct by providing 2 in-context examples for each of the 6 task categories. The original results of ReAct in their paper are produced based on Text-Davinci-002. However, this GPT version is no longer available, so we apply gpt-3.5-turbo-instruct instead to generate actions. For ExpeL, we directly take the guidelines from their appendix and append them to the ReAct agent at test time. For Reflexion, we generate reflective feedback based on the test rewards if the tasks fail. This reflective feedback is appended to the action generation prompt, immediately following the context-aware guidelines for AutoGuide + Reflexion. When there are multiple past episodes, we concatenate the reflection feedback from each episode. This process continues until the maximum number of episodes (or trials) is reached.

A.2.1.3 Implementation details of AutoGuide

We run the first 100 training tasks of ALFWorld to collect (τ_+, τ_-) pairs with ReAct+Reflexion and extract context-dependant guidelines on the collected data. For context identification, we provide 2-shot demonstrations for each of the 6 task categories. The corresponding prompt templates can be found in Appendix A.3. All parameter details are shown in Table A.1.

Parameter name	Value
Allowed Episode Length	50
n-shots	2
Agent Model	gpt-3.5-turbo-instruct
Context Identification Model	gpt-3.5-turbo-instruct
Guideline Selection Model	gpt-3.5-turbo-instruct
Guideline Extraction Model	gpt-4-1106-preview
Reflexion Model	gpt-4-1106-preview
top-k guideline selection	2

Table A.1: Experiment hyper-parameters on ALFWorld. The maximum allowed episode length and n-shots follow the same setup in ReAct.

A.2.2 WebShop [169]

A.2.2.1 Environment Details

WebShop provides an e-commerce environment, where the objective is to find and buy the product that matches the task-specific Instruction. The agent can select one of the following actions to perform:

- search[query]
- click[button]

Following ReAct, the agent can generate think actions to do planning or reflection. After buying a product, the environment returns a reward showing how well the bought product matches the target one in type, price, buying options, and attributes. The reward is calculated by:

$$r = r_{type} \cdot \frac{|U_{att} \cap Y_{att}| + |U_{opt} \cap Y_{opt}| + \mathbb{1}[y_{price} \leq u_{price}]}{|U_{att}| + |U_{opt}| + 1}$$

where y is the bought product and u is the target product. Same as ALFWorld, for WebShop, the agent takes (obs_t, act_t) pairs for every previous timestep t as input to generate the next action.

A.2.2.2 Baseline and Models

Following ReAct, experiments are done in a one-shot setting. We apply gpt-3.5-turbo-0613 to generate actions, but when the token number exceeds the token limit (for example, for the n-shot ReAct experiments in Table 2.1), we use the 16k version of gpt-3.5-turbo-0613 instead. For ExpeL, we could not find how many training tasks the framework used for training. Therefore, we directly apply the guidelines from the appendix of their paper at test time. We only consider ExpeL with guidelines, not ExpeL with in-context example selection in our experiments for a fair comparison. The in-context example selection method is orthogonal to our work and can be easily combined with our method. For Reflexion, as shown in their paper, their 2-shot Reflexion prompt does not work well on WebShop. Therefore, we re-write the prompt and apply gpt-4-1106-preview to generate episode-level reflections for all Reflexion experiments. Same as the experiments in ALFWorld, the generated reflections are appended to the action generation prompt, immediately following the context-aware guidelines for AutoGuide + Reflexion. In cases where multiple past episodes are available, the reflection feedback from each episode is concatenated. This process is repeated until the maximum number of trials is reached. Following Reflexion and ExpeL, the evaluation is done on 100 test tasks. The maximum number of allowed actions for each task is 15. At the same time, each search action shows the top 3 products for the search query. Please refer to Table A.2 for more details about the experiments.

A.2.2.3 Implementation details of AutoGuide

We randomly sample 50 training tasks from the training set of WebShop, on which we run ReAct+Reflexion to collect pairs and generate guidelines. The context identification prompt is one-shot, which is shown in Section A.3. At test time, we ask gpt-3.5-turbo-0613 to select each state's most relevant top 2 guidelines.

A.2.3 WebArena [188]

A.2.3.1 Environment Details

WebArena provides web-based benchmark environments that closely follow the data and functionality of real-world websites. Unlike other benchmarks like WebShop that provide

Parameter name	Value
Allowed Episode Length	15
# of Search Results	3
n-shots	1
Agent Model	gpt-3.5-turbo-0613
Context Identification Model	gpt-3.5-turbo-0613
Guideline Selection Model	gpt-3.5-turbo-0613
Guideline Extraction Model	gpt-4-1106-preview
Reflexion Model	gpt-4-1106-preview
top-k guideline selection	2

Table A.2: Experiment hyper-parameters on WebShop. The maximum allowed episode length, the number of search results per page, and n-shots follow the same setup in ReAct.

clean text of website information as observation, WebArena's webpage content is represented as an accessibility tree, which is a subset of the DOM tree with useful elements of a webpage. For our experiments, we focus on WebArena Reddit, which simulates the real Reddit websites with users, forums, and posts with abundant text information.

For each task in WebArena, the agent is expected to achieve a task-specific intent. At each timestep, WebArena provides a list of opened tabs, the accessibility tree of the focused webpage, and the URL of the current page as observation. For WebArena, each observation is long. Therefore, following the baseline in WebArena, at each timestep, we only provide the observation of the current timestep to the agent. We additionally provide up to 5 past actions for the agent to understand what it did in the past. The allowed actions in WebArena include the following:

- goto [url]
- click [element_id]
- type [element_id] [text] [1 for enter or 0 for not enter]
- press [key_combination]
- scroll [up or down]
- go_back

The maximum allowed number of actions for a single task is 20. Note that WebArena does not provide training tasks, but the work provides 19 demonstrations for Reddit, each of a different category. Therefore, we set these 19 tasks as the training tasks and then test on the rest 87 tasks.

A.2.3.2 Baseline and Models

We directly run the two-shot ReAct-style baseline in the official codebase of WebArena using gpt4-preview-1106. For ExpeL, the original paper does not include experiments on WebArena, therefore we try our best to implement our own version and run on the same training tasks as our method.

A.2.3.3 Implementation details of AutoGuide

We directly run ReAct on the tasks with τ_+ to collect τ_- actions and generate guidelines correspondingly. We provide a 5-shot prompt for the context identification module, which is shown in Figure A.3. At test time, the top 2 guidelines at each timestep are selected to guide action generation.

Parameter name	Value
Allowed Episode Length	20
n-shots	2
Agent Model	gpt-4-1106-preview
Context Identification Model	gpt-4-1106-preview
Guideline Selection Model	gpt-4-1106-preview
Guideline Extraction Model	gpt-4-1106-preview
top-k guideline selection	2

Table A.3: Experiment hyper-parameters on WebArena. The number of shots follows the same setup in ReAct.

A.2.4 Real Websites

A.2.4.1 Environment Details

We design a set of real-world website navigation tasks from 3 domains: Software Development (GitHub), Travel (Google Flights), and Education (Coursera), which have 30, 20, and 20 test tasks accordingly. Here are some example tasks:

- GitHub
  - Navigate to the repository for the Python library Seaborn with the most stars and show me all the open issues labeled with bug.
  - Go to the GitHub org for Spotify and open the pinned project with the most stars for me.

- Google Flights
  - Find me the one-way flight from Hong Kong to Paris departing on Oct 15th 2024 with the least emissions.
  - Show me the booking options of the one-way flight departing from Auckland on September 11, 2024, and arriving in Rome with the earliest departure time on that day.
- Coursera
  - Show me a Cybersecurity course that can finish within 1 month and show me all the reviews for the selected course.
  - Find me a Coursera guided project that covers Unity and show me its main page.

We follow the action space design of Visual WebArena [66], which has the following action types available:

- goto [url]
- click [element_id]
- hover [element_id]
- type [element_id] [text] [1 for enter or 0 for not enter]
- press [key_combination]
- scroll [up or down]
- tab_focus [tab_index]
- close_tab
- go_back
- go_forward

As the real websites are constantly and dynamically changing, we evaluate the completed task with human experts.

A.2.4.2 Baseline and Models

We directly run the two-shot SoM algorithm in the official codebase of Visual WebArena. The only modification we made from the original codebase is the bounding box detection algorithm, in which we further filter invisible bounding boxes from the list and add the list elements as interactable elements to consider.

A.2.4.3 Implementation details of AutoGuide

We provide a total of 6 training tasks (3 for GitHub, 2 for Google Travel, and 1 for Coursera), on which we collect human demonstration as τ_+'s and run SoM to collect τ_-'s. From the pairs with multimodal observations we generate text-based guidelines to guide action selection. Both context identification and guideline extraction are done by gpt-4-vision-preview, and we provide a 3-shot prompt for context identification. All parameter details are shown in Table A.4.

Parameter name	Value
Allowed Episode Length	15
n-shots	2
Agent Model	gpt-4-vision-preview
Context Identification Model	gpt-4-vision-preview
Guideline Selection Model	gpt-4-vision-preview
Guideline Extraction Model	gpt-4-vision-preview
top-k guideline selection	2

Table A.4: Experiment hyper-parameters on multimodal real-world website tasks.

A.3 Prompt Templates

A.3.1 Context Identification

We present our prompt templates for the context identification module $\mathcal{M}_{\text{context}}$ (Equation (2.1)) for WebShop, ALFWorld, and WebArena in Figures A.1 to A.3, respectively. In ALFWorld, there exist six categories of tasks, and we use context identification prompting with 2-shot examples for each task, following the practice by [170]. Figure A.2 shows one example for the `pick_and_place` tasks.

A.3.2 Guideline Extraction

Figures A.4 to A.6 detail our prompt templates $\mathcal{M}_{\text{guideline}}$ for extracting guidelines (Equation (2.2)) in the WebShop, ALFWorld, and WebArena domains.

A.3.3 Context Matching

Figure A.7 shows our prompt template for matching the generated context with one of the existing contexts if there is any similar context, for the construction of the set of context-aware

Context Identification Prompt for WebShop

Instruction

Now, you'll get a snippet of a trajectory. Your job is to generate a brief and general summarization of the current status. Keep it broad and general, avoid any information about specific instructions, products, or buying options. For different 'Instructions' with the same status, the summarization should be the same.

In-Context Example

Webshop
Instruction:
i would like a bottle of bright citrus deodorant under 50 dollars
[Search]
SUMMARIZATION: The assistant is on the search page with a [Search] box.

Action: search[bright citrus deodorant]
Observation:
[Back to Search]
Page 1 (Total results: 3)
[Next $>$]
[B078GWRC1J]
Bright Citrus Deodorant
$10.99
[B078GTKVXY]
Ginger Fresh Deodorant
$10.99
[B08KBVJ4XN]
Barrel and Oak - Aluminum-Free Deodorant
$15.95
SUMMARIZATION: The assistant is on the search result page with a list of products.

Action: click[B078GWRC1J]
Observation:
[Back to Search]
[$>$ Prev]
scent [assorted scents][bright citrus][ginger fresh]
size [travel set (4-pack)][(pack of 1)][3-ounce (2-pack)]
Bright Citrus Deodorant
Price: $10.99
Rating: N.A.
[Description]
[Features]
[Reviews]
[Attributes]
[Buy Now]
SUMMARIZATION: The assistant is on a product's main page.

Action: click['bright citrus' '(pack of 1)']
Observation: Invalid Action!
SUMMARIZATION: The assistant failed to click on a button and got 'Invalid Action!'.

Action: click[bright citrus]
Observation: You have clicked bright citrus.
SUMMARIZATION: The assistant has successfully clicked on a button.

Action: click[Description]
Observation:
[Back to Search]
[$<$ Prev]
Bright Citrus Deodorant for sensitive skin
SUMMARIZATION: The assistant clicked on and navigated to a product's subpage(one of [Description], [Features], [Reviews], and [Attributes]).

Action: click[Buy Now]
Observation: Invalid Action!
SUMMARIZATION: The assistant failed to click on a button and got 'Invalid Action!'.

Action: search[bright citrus deodorant under 50]
Observation: Invalid Action!
SUMMARIZATION: The assistant failed to search on the page and got 'Invalid Action!'.

Input

Now it's your turn:
{Current trajectory}

Figure A.1: Our prompt template for context identification (Equation 2.1) in the WebShop domain.

guidelines (Section 2.3.2) and the retrieval of relevant guidelines during testing (Section 2.3.3) in all the three domains: WebShop, ALFWorld, and WebArena.

A.3.4 Guideline Selection

For selecting only k most relevant guidelines in case there are more corresponding context-aware guidelines during testing (Equation 2.3), we use the prompt with Figure A.8 for WebShop and ALFWorld and Figure A.9 for WebArena.

Context Identification Prompt for ALFWorld

Instruction

You'll get a snippet of a trajectory of an text-based ALFRED task. Your job is to generate a brief and general summarization of the current status following 'SUMMARIZATION: '.
Keep it broad and general, avoid any information about specific objects and locations.
Here are two examples:

In-Context Example

You are in the middle of a room. Looking quickly around you, you see a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 1, a diningtable 3, a diningtable 2, a diningtable 1, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a sidetable 1, a sinkbasin 1, and a toaster 1.
Your task is to: find some apple and put it in sidetable.
> think: To solve the task, I need to find and take an apple, then put it in sidetable.
OK.
> think: First I need to find an apple. An apple is more likely to appear in fridges (1), diningtables (1-3), sidetables (1), countertops (1), sinkbasins (1), garbagecan (1). I can check one by one, starting with fridge 1.
OK.
> go to fridge 1
The fridge 1 is closed.
SUMMARIZATION: Looking for an object here, but it is closed.
> open fridge 1
You open the fridge 1. The fridge 1 is open. In it, you see a lettuce 2, a mug 2, a potato 2, and a tomato 1.
SUMMARIZATION: Looking for an object, but it's not here.
> go to table 1
Nothing happens.
SUMMARIZATION: Tried to go to a location but failed.
> go to diningtable 1
On the diningtable 1, you see nothing.
SUMMARIZATION: Looking for an object, but it's not here.
> go to garbagecan 1
On the garbagecan 1, you see a apple 3, and a egg 3.
SUMMARIZATION: Looking for an object and found the desired object here.
> think: Now I find an apple (3). Next, I need to take it.
> take apple 3 from garbagecan 1
You pick up the apple 3 from the garbagecan 1.
SUMMARIZATION: Successfully took an object.
> think: Now I find an apple (3). Next, I need to put it in/on sidetable 1.
OK.
> go to sidetable 1
On the sidetable 1, you see nothing.
SUMMARIZATION: Looking for a location to put an object in/on.
> put apple 3 in/on sidetable
Nothing happens.
SUMMARIZATION: Tried to put an object in/on a location but failed.
> put apple 3 in/on sidetable 1
You put the apple 3 in/on the sidetable 1.
SUMMARIZATION: Successfully put an object in/on a location.

In-Context Example

...

Input

Now it's your turn:
{Current trajectory}

Figure A.2: Our prompt template for context identification (Equation 2.1) in the ALFWorld domain.

A.4 Example Context-Aware Guidelines

In Figures A.10 and A.11, we show a list of possible contexts and context-aware guidelines on WebArena and real-world websites, respectively.

Context Identification Prompt for WebArena

Instruction

You are an autonomous intelligent agent tasked with navigating a web browser. You will be provided with the following information:

1. A list of context summarizations you have seen in the past.
2. A snippet of the current web page's accessibility tree: a simplified representation of the webpage with key information and the current web pages' URL.

Please generate the summarization of the current observation after 'SUMMARIZATION:'.
Here are some requirements:
Requirement 1: Different types of webpages should have clearly different summarizations. For example, for GitHub there can be the main page of GitHub, the overview page of a GitHub user, the issues page of a GitHub repository, the search result page of GitHub. etc, you should clearly categorize those and make sure not to mix them up.
Requirement 2: Important: The summarization should be general, and concise, without any user/object/task specific information, instead, websites that fall into the same categories should have the same summarization, for example, the main page of every reddit forum should be categorized as the same context summarization: On the main page of a Reddit forum. You should never include the specific name of the forum in the summarization.
Requirement 3: The URLs will be very useful for you to determine the summarization.
Requirement 4: If the context is the same as one from the seen list, directly copy the best matching one word by word.

In-Context Example

Observation:
The current web page's accessibility tree:
Tab 0 (current): Postmill
[1] RootWebArea 'Postmill' focused: True
    [31] HeaderAsNonLandmark ''
        [32] link 'Home'
    [55] link 'Forums'
    [56] link 'Wiki'
    [64] searchbox 'Search query'
    [65] link 'Notifications (0)'
    [66] link 'Submit'
    [12] button 'MarvelsGrantMan136' focused: True hasPopup: menu expanded: True
    [243] link ' Profile'
    [239] link ' My account'
    [245] link ' User settings'
    [255] link ' Block list'
    [286] separator '' orientation: horizontal
    [270] button ' Dark mode'
URL: http://reddit.com/
SUMMARIZATION: On the Reddit main page.

In-Context Example

...

Input

Here are the inputs:
{Seen list}
{Current trajectory}

Figure A.3: Our prompt template for context identification (Equation 2.1) in the WebArena domain.

Guideline Extraction Prompt for WebShop

Instruction

{Task description}. You will be provided with a desired and undesired trajectory of the same task. What is the first action that differs between the two trajectories? Why do you think it makes one trajectory failed and the other successful? Based on your answer, generate an action guideline to make future task avoid the same mistake. The guideline should specify what to do in what situation in the format of "When in what status, you should (or should not)...". On a product's page with product information, strictly refer to the option buttons as 'buying options such as sizes, colors, scents, and flavors', and clearly say that buying options are not subpages like [Description] and [Attributes] when you mention buying options. Your guideline must be general enough for any task, therefore never include any task-specific information, instead, refer to all the requirements as the requirements in Instruction. Strictly follow what the desired trajectory does and never suggest actions that the desired trajectory didn't do. When referring to actions, use the allowed action format. You should make your answer concise, limit your answer within 256 tokens, and put your answer in this format: 'Reasoning: ...
Guideline: ...'.

Input

Desired Trajectory:
{Desired trajectory}
Undesired Trajectory:
{Undesired trajectory}

Figure A.4: Our prompt template for guideline extraction (Equation 2.2) in the WebShop domain.

Guideline Extraction Prompt for ALFWorld

Instruction

{Task description}. You will be provided with a desired and undesired trajectory of the same task. What is the first action that differs between the two trajectories? Why do you think it makes one trajectory failed and the other successful? Based on your answer, generate an action guideline to make future task avoid the same mistake. The guideline should specify what to do in what situation in the format of "When in what status, (optional: if you want to ...) you should (or should not)... (optional: a short example for demonstration.)". For the 'When in what status' part, directly use the words in SUMMARIZATION.
Here are two examples:
Example 1: When looking for an object, if you want to find a kitchen-related object like a spatula, you should start from the most possible locations.
Example 2: When looking for an object and found the desired object at the location, You should only take the exact object that you want.
Strictly follow what the desired trajectory does and never suggest actions that the desired trajectory didn't do. When referring to actions, use the allowed action format. You should make your answer concise, limit your answer within 128 tokens, and put your answer in this format: 'Reasoning: ... Guideline: ...'.

Input

Desired Trajectory:
{Desired trajectory}
Undesired Trajectory:
{Undesired trajectory}

Figure A.5: Our prompt template for guideline extraction (Equation 2.2) in the ALFWorld domain.

Guideline Extraction Prompt for WebArena

Instruction

{Task description}
You just finished a task but failed. For this failed task, we provide a human demonstration for you. Please compare the demonstration with your generated action at each step, reason about the intention of the correct action, and then generate an action guideline for future tasks to avoid the same mistake and make the future tasks successful.

Here's the information you'll have:
1. The current observation:
* The task objective: the task you're trying to complete.
* The current web page's accessibility tree: a simplified representation of the webpage, providing key information.
* The current web page's URL: the link of the page you're currently on.
* The open tabs: the tabs you have opened.
* The previous actions: a sequence of past actions that you performed.

2. The action you generated in the failed run.

3. The correct action that you should take.

4. Demonstration actions in later steps on the same page.

Based on the information, please generate a short and concise guideline that guide you to issue the correct action.
Important: The guideline should be general enough to generalize to all similar tasks, not only this task. Therefore do not include any task-specific information in your guideline, for example a user name, a specific forum, the specific text you want to enter, or any number ID in [] in front of each element, for example [123], the numbers are randomly generated therefore never include them in your guideline. However, for other non-specific elements like "link 'Forums'" or "button 'Create submission'", you should specifically include them in the exact text in your guideline.
When referring to a url in your guideline, specify it as detailed as possible, only replace the task specific information as a palceholder, for example, replace a forum iphone with <forum_name> and specify the url in full, starts with http://.
The guideline should be less than 128 tokens.
Please refer to "the previous actions" and "Demonstration actions in later steps" to generate more accurate descriptions of your purpose and the sequence of actions to achieve the purpose. make sure to emphasize the order of the actions, do not miss any single action, and put them in 1. 2. 3. ..., for example 'after you typed in all the text, you should do these sequentially: 1. ..., 2. ... . You must strictly follow the order."
When you mention multiple steps of actions, also mention in the guideline that you should refer to the PREVIOUS ACTIONS to reason about which actions you did and what you should do next. Specify that you should not repeatedly issue the same action, but should move on to the next action instead.
Only specify what to do or what not to do, don't explain why.
It is important to clearly specify when to issue a stop action when the stop action is either the correct action or in the 'Demonstration actions in later steps on the same page.', do not specify the 'answer' in 'stop [answer]' because answer is different for different tasks, and do not mention anything about stop if this action is neither in "The correct action that you should take." nor "Demonstration actions in later steps on the same page.".
Please strictly adhere to the 'correct action that you should take', do not propose other actions.

Input

Here are the information you need:
{Observation}
The action you generated in the failed run:
{Predicted action}
The correct action that you should take:
{Demo action}
Demonstration actions in later steps on the same page:
{Later action}
Please put your answer in this format: Reasoning: ... Guideline: ...

Figure A.6: Our prompt template for guideline extraction (Equation 2.2) in the WebArena domain.

Context Matching Prompt

Instruction

{Task description}
A task trajectory can be long. Therefore the assistant summarizes the status of each step.
For different task with the same status, the summarization should be the same, therefore please ignore any information about instructions or products.
You will be provided with the following:
1. A list of summarizations the assistant saw in the past.
2. A newly generated summarization.
Please determine if any summarization from the list matches the exact same status as the newly generated one. If yes, answer the index of the corresponding summarization, for example "Answer: 2"; otherwise, "Answer: None".

Input

Seen Summarizations:
{List of contexts}

Figure A.7: Our prompt template for context matching (Section 2.3.3) in all the WebShop, ALFWorld, and WebArena domains.

Guideline Selection Prompt for WebShop and ALFWorld

Instruction

{Task description}. You will be equipped with the following resources:
1. A list of action guidelines with valuable guidelines.
2. Trajectory history, which includes recent observations and actions.
Not all guidelines are useful to generate the next action. Please select the guidelines that are useful and relevant to the next action given the trajectory and recent observations. To generate the next action, which guidelines from the provided guidelines are most useful to directly tell you what to do for the next action? You can select up to 2 guidelines, and put the indices of the selected guidelines in a python list. For example if you select guideline 1, 5, answer: [1, 5]. If none of them are useful for generating the next action, answer the empty list [].

Input

{List of guidelines}
{Current trajectory}

Figure A.8: Our prompt template for selecting the most relevant context-aware guidelines at test time (Equation 2.3 from Section 2.3.3) in the WebShop and ALFWorld domains.

Guideline Selection Prompt for WebArena

Instruction

{Task description}. At each time step, you need to generate one action given the current observation.
You will be equipped with the following resources:
1. A list of action guidelines with valuable guidelines.
2. The intent of the task, which is the objective/goal that you should achieve.
3. Trajectory history, which includes the current observation and a sequence of past actions.
Not all guidelines are useful to generate the next action. Please select the guidelines that are useful and relevant to the next action given the current observation and past actions.
To generate the next action, which guidelines from the provided guidelines are most useful to directly tell you what to do for the next action? You can select 3 guidelines (or less if there are less than 3 guidelines), and put the number indices of the selected guidelines in a python list. For example if you want to select guideline 2 and 5, answer [2, 5]. If none of them are relevant, answer [].

Input

{List of guidelines}
{Current trajectory}

Figure A.9: Our prompt template for selecting the most relevant context-aware guidelines at test time (Equation 2.3) in the WebArena domain.

Context: On the Reddit main page.
Context-Aware Guideline:
- When on the Reddit main page, if you want to change your bio, you should click on the "link 'Profile'", which is located in the user menu dropdown, right above the "link 'My account'". The correct action format to do this is ```click [profile_link_id]```.
- When on the Reddit main page, if you want to navigate to a specific forum, you should click on the "link 'Forums'", which is located at the early part of the observation, right above the link 'Wiki'. The correct action format to do this is ```click [link_id]```.
- When on the Reddit main page, if you want to create a new forum, you should click on the "link 'Forums'", which is located near the top of the observation, right above the "link 'Wiki'". The correct action format to do this is ```click [link_id]```.
- When on the Reddit main page, if you want to like all submissions created by a specific user in a specific subreddit, you can directly navigate to the user's page with the action format ```goto [url]```, replacing [url] with the user's page URL, formatted as http://<platform_domain>/user/<username>.

Context: On the overview page of a Reddit user.
Context-Aware Guideline:
- When on the overview page of a Reddit user, if you want to interact with submissions from a specific subreddit, you should first navigate to the 'Submissions' tab to filter the content by the user's submissions. The correct action format to do this is ```click [link_id]```, where [link_id] is the ID of the 'Submissions' link in the main content area of the page.

Context: On the biography edit page of a Reddit user.
Context-Aware Guideline:
- When on the biography edit page of a Reddit user, if you want to change the biography text, you should do these sequentially: 1. Click on the textbox 'Biography' to focus it, located in the main section of the page, with action ```click [textbox_id]```. 2. Select all text inside the textbox using the action ```press [Meta+a]```. 3. Clear the selected text with the action ```press [Backspace]```. 4. Type the new biography content into the textbox 'Biography' with action ```type [textbox_id] [new_content] [1]```. 5. Click on the button 'Save', located below the textbox 'Biography', to submit the changes with action ```click [button_id]```. After these steps, issue a stop action when the task is complete.

Context: On the page listing all forums on a Reddit-like platform.
Context-Aware Guideline:
- When on the page listing all forums on a Reddit-like platform, if you want to navigate to a specific forum, you should do these sequentially: 1. click on the "link 'Alphabetical'", which is located in the main area, to sort forums alphabetically. The correct action format to do this is ```click [link_id]```. 2. After the forums are sorted, click on the specific "link '<forum_name>'" that you wish to navigate to. The correct action format to do this is ```click [link_id]```.

Context: On the Reddit page to create submission.
Context-Aware Guideline:
- When on the Reddit page to create submission, if you have filled in the 'Title' and 'Body' textboxes but do not see the button 'Create submission', you should scroll down to reveal more of the page. The correct action format to do this is ```scroll [down]```. After scrolling, if you want to submit the post, you should click on the button 'Create submission', which is located after the 'Body' textbox. The correct action format to submit the post is ```click [button_id]```.

Context: On the main page of a Reddit forum.
Context-Aware Guideline:
- When on the main page of a Reddit forum, if you want to find posts related to a specific topic among the top posts, you should first sort the posts by their popularity to ensure you are viewing the most relevant content. To do this, you can: 1. click on the "button 'Sort by: Hot'", which is located in the main section of the page, below the forum heading and above the first article. The correct action format to do this is ```click [id]```. 2. After the sorting options have expanded, click on the "link 'Top'", which will appear as a new option under the sorting button. This action should be repeated once. The correct action format to do this is ```click [id]```.
- When on the main page of a Reddit forum, if you want to create a new post, you should click on the "link 'Submit'", which is located in the early part of the observation, right below the "link 'Notifications (0)'". The correct action format to do this is ```click [link_id]```.

Context: On the submissions page of a Reddit user.
Context-Aware Guideline:
- When on the submissions page of a Reddit user, if you want to perform an action on specific subreddit submissions but do not see them, you should scroll down to reveal more submissions. The correct action format to do this is ```scroll [down]```. After scrolling, if you find a submission from the desired subreddit, such as 'UpliftingNews', and need to downvote it, click on the button 'Downvote' located at the end of the submission's details. The correct action format to downvote is ```click [downvote_button_id]```. Once all required actions are performed on the submissions, issue a stop action with the format ```stop []```.

Context: On the submissions page of a Reddit forum sorted by top.
Context-Aware Guideline:
- When on the submissions page of a Reddit forum sorted by top, if you want to review posts but see "There's nothing here...", you should expand the time range to view more posts. Do this by clicking on the button 'From: Past 24 hours', which is located in the main section of the page, below the heading '/f/books' and above the StaticText "There's nothing here...". The correct action format to do this is ```click [button_id]```. After expanding the time range, if you find a post that meets the task criteria, issue a stop action.

Context: On the submissions page of a Reddit forum sorted by new.
Context-Aware Guideline:
- When on the submissions page of a Reddit forum sorted by new, if you want to upvote the newest post and you have already clicked the upvote button for the first article entry, you should issue the stop action. The correct action format to do this is ```stop```.

Context: On the page of a Reddit post.
Context -Aware Guideline:
- When on the page of a Reddit post, if you have already navigated to the 'Submit' link, filled in the image URL and title, and clicked the 'Create submission' button, you should consider the task complete. The correct action format to do this is ```stop```.

Figure A.10: Example contexts and corresponding guidelines for WebArena.

Context: On the main page of GitHub.
Context-Aware Guideline:
- When on the main page of GitHub, if you want to search for a repository, you should type the search query into the search bar at the top of the page, which is visually identifiable and typically labeled with text like 'Search or jump to...'. Do not type into any other input fields. Perform the action as follows: ```type [search bar id] [search query] [1]```.

Context: On the issues page of a GitHub repository.
Context-Aware Guideline:
- On the issues page of a GitHub repository, if you want to filter issues by a specific label, you can type the label filter in the search input field, which is usually at the top of the current webpage and shown as the first appeared "[INPUT] []" in observation. The correct action format to do this is ```type [id] [label:"specific_label"] [1]```. After applying the filter: 1. Click on the link that shows the number of closed issues, which is labeled as "[A] [num Closed]" in observation, located near the search input field. The correct action format is ```click [link_id]```.

Context: On the search result page of GitHub.
Context-Aware Guideline:
- On the search result page of GitHub, if you want to filter the search results to find a specific organization, after you have typed the organization's name in the search bar, you can do these sequentially: 1. click on the "[LI] [Users]" in the left sidebar, which is visually represented by the blue text "Users" and is the first "[LI] [Users]" in observation, by action ```click [li_id]``` 2. if the user or organization filter is applied, click on the organization's name, which is visually represented by the blue text "Meta" under the "Users" section, by action ```click [link_id]```

Context: On the search result page of Coursera.
Context-Aware Guideline:
- On the search result page of Coursera, if you want to select a course, after you have typed the course topic in the search bar, you can do these sequentially: 1. click on the filter for courses, which is represented by "[INPUT] []" and visually located in the filter section on the left side of the webpage, by action ```click [input_id]``` 2. click on the course link, which is represented by "[A] [course_name]" and visually located in the main content area of the webpage, by action ```click [link_id]```. This action should be repeated twice. 3. switch to the new tab that contains the course details by ```page_focus [1]```.

Context: On the main search page of Google Flights.
Context-Aware Guideline:
- On the main search page of Google Flights, if you want to search for a one-way flight with specific departure and destination airports and date, you can do these sequentially: 1. type the departure airport code in "[INPUT] [Where from?]", which is the first input box at the top of the search area, by action ```type [element_id] [airport_code] [1]``` 2. type the destination airport code in "[INPUT] [Where to?]", which is next to "[INPUT] [Where from?]", by action ```type [element_id] [destination_airport_code] [1]``` 3. click on "[DIV] [Round trip]" to change the trip type, located at the top of the search area, by action ```click [element_id]``` 4. click on "[LI] [One way]" to select the one-way trip option, which appears after clicking "[DIV] [Round trip]", by action ```click [element_id]``` 5. type the departure date in "[INPUT] [Departure]", which is next to "[INPUT] [Where to?]", by action ```type [element_id] [date] [1]``` 6. click on "[BUTTON] [Done]" to confirm the date, which appears after entering the departure date, by action ```click [element_id]``` 7. click on "[BUTTON] [Search]" to perform the flight search, which is below the search fields, by action ```click [element_id]```

Context: On the search result page of Google Flights with a list of Best departing flights and Other departing flights.
Context-Aware Guideline:
- On the search result page of Google Flights with a list of Best departing flights and Other departing flights, if you want to select a flight, you can click on the first flight option under the "Best flights" section. The correct action format to do this is ```click [flight_option_id]```, where [flight_option_id] is the id of the [LI] element corresponding to the first flight listed. This [LI] element is visually located at the top of the list of flights and contains the departure and arrival times, airline name, flight duration, and other details.
- On the search result page of Google Flights with a list of Best departing flights and Other departing flights, if you want to locate the flight with the least emissions, you can do these sequentially: 1. Click on the sort options button, which is visually located at the top of the flight list and shown as "[BUTTON] [Sort by:]" in observation, by action ```click [sort_button_id]```. 2. Then, click on the emissions sort option, which is visually located in the sort options dropdown and shown as "[LI] [Emissions]" in observation, by action ```click [emissions_option_id]```. 3. Finally, click on the first flight listed under the "Best flights" section, which is visually located at the top of the list and shown as "[LI] [flight_details]" in observation, by action ```click [first_best_flight_id]```. Repeat this action twice.

Figure A.11: Example contexts and corresponding guidelines for real-world websites.

APPENDIX B

Beyond Blind Following: Evaluating Robustness of LLM Agents under Imperfect Guidance

B.1 Broader Impacts

This work introduces a benchmark for evaluating the robustness of language agents in the presence of imperfect guidance. We encourage its use as a complementary resource within broader research efforts on verification, interpretability, and agent alignment. However, results from controlled settings should not be overinterpreted when applying agents to real-world scenarios. Benchmarks alone cannot guarantee safety or reliability. Agents trained or evaluated using this benchmark may still misinterpret flawed guidance or fail to recover from subtle inconsistencies. Additional safety mechanisms, human supervision, and verification procedures may be necessary to ensure responsible deployment. While our benchmark can support the development of more robust agents in applications such as web automation, digital assistance, and interactive education, we strongly discourage its use for building systems that pursue harmful or deceptive objectives. We hope this chapter contributes to the advancement of agents that are not only capable but also trustworthy and adaptable.

B.2 Environment Details

Our benchmark includes two components, MIRAGE-World and MIRAGE-Web, which provide a comprehensive testbed for evaluating the robustness of language agents under imperfect guidance. MIRAGE-World consists of procedurally generated tasks across navigation, cooking, and gaming domains, with full control over task complexity, layout, and injected imperfections. MIRAGE-Web is built on realistic websites from WebArena [188] and includes long-horizon browser-based tasks in shopping, reddit, shopping admin, and gitlab. Across both components, we introduce imperfections at two levels: changes to the guidance itself, such as missing steps or ambiguous descriptions, and changes to the environment, such

as altered object placements, seasonal differences, or dynamic obstacles. All tasks remain solvable by design, and agents must learn to detect, reason through, and recover from these deviations. In the sections that follow, we describe each environment in detail, including task structure, observation and action formats, and the guidance generation process. At the same time, all environments in MIRAGE are procedurally generated or derived from WebArena, which contains no personally identifiable or offensive content. We verified that no task data or webpages include such material. The released artifacts will include detailed instructions to facilitate full reproducibility of our experiments. All materials will be made publicly available under the MIT License, allowing free use, modification, and distribution for research purposes. Any third-party resources (e.g., datasets or pretrained models) used in this chapter will be properly cited and linked in the release documentation.

B.2.1 MIRAGE-World Navigation

B.2.1.1 Task Overview

In this task, the agent must navigate a procedurally generated grid environment to reach a destination. All experiments presented in this chapter use a 12×12 map. The agent starts at a random location within the farthest 25 percent of traversable tiles from the goal. At each timestep, it receives a partially observable 3×3 view centered on its current position, along with its current facing direction (for example, north). Each tile in the map corresponds to a 6×6 unit region. Therefore, the agent gets an 18×18 unit visual field at each timestep.

Observation Space At each timestep, the agent receives a 3×3 first-person observation centered on its current position, along with the direction it is facing (for example, *Direction: You are facing north*). Observations are presented in one of two formats:

- **Encoded language:** Each tile within the observation window is described using natural language. For example: *"There is a group of green pines on the front-right and right tiles."*
- **Matrix format:** The observation is rendered as an 18×18 grid corresponding to the 3×3 tile window. Each tile is labeled with an object type or ID (for example, `road`, `house_2`). Object IDs such as `tree_2` are used to distinguish between different instances of the same type.

Action Space The agent can perform the following actions:

- `forward`: The agent moves forward by one tile in its current facing direction. Tiles such as grass, trees, houses, and other obstacles are non-traversable.
- `turn left`, `turn right`, `turn around`: Rotate in place.

Reward The task ends either when the agent reaches the tile adjacent to the destination or when the episode exceeds a maximum step limit. The task is considered successful if the agent reaches the destination. The agent receives a reward at the end of the episode, computed as

$$\text{reward} = 1 - \frac{\text{Distance to destination}}{\text{Maximum possible distance}} \tag{B.1}$$

with the distance measured in number of tiles.

Environment Generation Maps are procedurally generated by arranging patches of grassland along a diagonal axis from the upper-left to the lower-right corner of the grid. Each patch is separated by a one-tile-wide road, which ensures that all areas remain globally connected. After the terrain layout is created, a destination is placed on a grass tile adjacent to a road. Next, objects such as trees and houses are added within the grass patches according to predefined spacing constraints. Finally, the agent is placed on a road tile from the farthest 25 percent of the map relative to the destination.

B.2.1.2 Guidance Generation

Perfect Guidance A BFS planner is run on the map to generate an optimal path to the destination. This path is then translated into three variants of natural language guidance:

- Tile-based directional guidance. This format provides precise step-by-step instructions using tile distances and turning directions. The agent can follow the guidance directly to reach the destination without needing to interpret its surroundings. Example:

  Start! You are currently facing east.
  Step 1. Turn left to face north.
  Step 2. Move forward 3 tiles (north).
  Step 3. Turn left to face west.
  Step 4. Move forward 2 tiles (west) to approach the destination.

- Turn-based navigation cues. This format omits exact movement distances and instead provides directional instructions focused on when to turn. The agent must rely on the observed road layout to decide the appropriate moment to execute each turn. Example:

Start! You are currently facing east.
Step 1. Turn left until you are facing north.
Step 2. Continue going north; turn left at the first intersection.
Step 3. Go straight. Your destination is just ahead, before any intersections.

- Observation-anchored instructions. This format relies on visual descriptions of the environment. The agent must monitor its 3×3 observation window to recognize when the described scene appears and then decide how to act based on that context. Example:

  Start! You are currently facing east.
  Step 1. Turn left until you are facing north.
  Step 2. Walk forward (toward north) until you see:
  There is a group of green pines on the back-right tile.
  There is a house on the front-right tile.
  The front-left, left, right, and back tiles are roads that you can step on.
  Step 3. Turn left until you are facing west.
  Step 4. Walk forward (toward west) until you are close to the destination.

While all three formats are sufficient to complete the task, they differ in the level of reasoning and environment perception required. The first variant is fully observation-agnostic and specifies an exact action sequence that the agent can follow without interpretation. In contrast, the latter two require the agent to interpret its surroundings during execution and adapt its actions based on local context.



Figure B.1: Four navigation environment variants. From left to right: (1) Standard — typical terrain layout; (2) Seasonal — snowmen and ice blocks appear, flowers and mushrooms disappear, and trees become snow-covered; (3) Construction — an intersection is blocked to simulate roadwork, requiring the agent to find a detour; (4) Dynamic — a pedestrian moves toward the agent, and decorative lamps change color at each timestep.

Imperfect Guidance We implement several types of imperfect guidance to introduce realistic forms of noise and variability. These imperfections fall into two broad categories. The first involves flaws in the guidance itself. The second arises from changes in the environment

that make otherwise accurate guidance less applicable. Such changes may be long-term, including seasonal transitions, terrain alterations, or urban development, or short-term, such as moving pedestrians, passing vehicles or dynamic visual signals. Examples are shown in Figure B.1.

1. Noisy Directions This variant introduces small perturbations to the first two types of perfect guidance: tile-based directional instructions and turn-based navigation cues. It alters movement counts to simulate realistic cases where numerical references may be slightly inaccurate. In everyday communication, for instance, a person might say "walk forward for about 200 meters" or "go past two intersections", even if the actual distance differs. Such imprecision is common, as people often rely on estimates or memory, leading to subtle inconsistencies in the guidance they give. To emulate this, we randomly apply a $+1/-1$ offset to movement tiles or turns for some steps in the perfect guidance. Notice that these changes only affect the guidance text, not the environment itself. As a result, the original path to the destination is still valid, and the task remains fully solvable.

The environment naturally provides feedback that helps the agent recognize imperfections in guidance. For example, if the agent turns too early, it may encounter non-traversable terrain such as trees or houses. Likewise, if instructed to "walk past three intersections" but only two are visible before reaching the map boundary, the agent can infer a mismatch. To support recovery more explicitly, we incorporate observation-based hints into the guidance itself. This mirrors how humans often rely on contextual clues and use visual markers to verify progress. For instance, a person might say, "Walk about 200 meters until you see a tower, then turn left." Even if the distance estimate is inaccurate, the tower provides a clear signal for when to act. Similarly, our guidance includes visual anchors, such as: "Turn left at the first opportunity when you see a house on the front-right tile." These observation-grounded phrases act as external checkpoints, helping agents identify potential imperfections in the guidance based on what they see in the environment.

2. Seasonal Mismatch This setting introduces a mismatch between the guidance and the current environment's visual appearance. The agent receives a guide written for a summer map but must operate in a winter version of the environment. Shared landmarks such as roads and houses remain unchanged. However, many other visual elements differ: for example, trees become snow-covered, and new seasonal objects such as snowmen or ice blocks are introduced. Some examples include:

- Summer-only: flowers, mushroom

- Winter-only: snowman with a hat
- Transformed objects: green oak tree → snow-laden oak tree

Since the layout of the grasslands and roads remains unchanged, the original path to the destination is still valid, and the task remains fully solvable.

3. Construction Blockages We simulate temporary road closures by inserting construction blocks, such as barricades or cones, along the originally optimal path. To make these disruptions easier for agents to detect and reason about, all construction blocks in this setting are placed as uniform 2×2 regions. After insertion, we perform a lightweight solvability check to ensure that at least one valid path to the destination still exists. In most cases, this check passes without issue due to the connectivity of the map. In the rare case that no valid path remains, we either relocate the construction block or regenerate the map. As a result, this imperfection realistically captures scenarios such as roadwork or temporary closures, where agents must recognize unexpected obstacles and plan a detour while still completing the task.

4. Dynamic Elements In contrast to long-term environmental changes such as seasonal shifts, dynamic elements represent short-term variations that occur within a few timesteps. These include visual or physical disruptions that change over time, such as flashing decorative lamps or moving pedestrians. We introduce two forms of dynamics into the environment:

- Color-changing lamps: Decorative roadside lamps cycle through different colors every timestep.
- Moving pedestrians: Human figures occasionally walk into the agent's path, temporarily blocking forward movement before continuing on their route.

These dynamic elements can mislead agents in subtle but significant ways. For example, guidance extracted from videos or step-by-step tutorials may instruct, "Turn right when you see three blue lamps". If the lamps change color and appear green when the agent arrives, it may become confused or act prematurely. Similarly, if told to "walk forward three tiles" but blocked by a moving pedestrian after the second tile, the agent may incorrectly assume it has advanced. This type of imperfection exposes a key limitation: language agents often over-rely on static, snapshot-based guidance and lack the situational awareness needed to adapt to dynamic environments. By introducing these short-term mismatches, we evaluate whether agents can pause, reassess, and manage small inconsistencies, which is essential for reliable behavior in real-world settings.

B.2.2 MIRAGE-World Cooking

B.2.2.1 Task Overview

In this task, the agent takes on the role of a chef working in a multi-location cooking environment. The objective is to prepare and serve dishes based on customer orders. To do so, the agent must navigate between four key locations: the restaurant, where cooking occurs, as well as the store, the farm, and several harbor sites. Some ingredients are available at the restaurant, while others must be gathered from these external sources. Once all required ingredients are collected, the agent returns to the kitchen to prepare and serve the requested dishes.

B.2.2.2 Task Generation

Each cooking task is generated by sampling one or more dish templates from a predefined recipe set. A dish template specifies required ingredients, optional additions, preparation steps, and seasoning constraints. For example, a `taco` may require a tortilla as a base, with optional ingredients such as shrimp or beef included depending on the task instance. Each sampled dish is also assigned a flavor profile, such as `Spicy Citrus`, which is associated with up to four seasonings. Ingredients are linked to specific preparation methods; for instance, chicken may be pan fried, deep fried, or grilled and then chopped for use in a fried rice recipe.

Once selected, all ingredients and seasonings are distributed across resource locations following semantic constraints. For example, seafood is primarily available through fishing at harbor sites, though some types may also be found at the store. The ingredient pool includes six farm-harvestable crops, twenty-two seafood varieties, five types of meat, sixteen store-exclusive items, and twenty-six seasonings. The benchmark currently supports five core dish templates with varying preparation complexity and ingredient diversity: `sushi`, `fried rice`, `salad`, `baked seafood`, and `taco`. Seasoning rules are enforced at execution time. Seasonings designated for use during cooking must be applied while ingredients are in tools such as pans or grills, whereas those intended for after cooking can only be added once ingredients are plated.

While the current benchmark emphasizes the effects of imperfect guidance, the framework is designed to be extensible along multiple dimensions. It supports the addition of new dish templates and resource locations with minimal modification. More complex configurations, such as time-sensitive processes or multi-agent coordination, can also be incorporated to further increase task difficulty.

B.2.2.3 Observation Space

At each timestep, the agent receives a natural language observation that reflects the result of its most recent action, along with updated information about the current environment state. This includes changes such as cooking progress and tool usage at the agent's current location. Below are examples of observations under different conditions:

- **Change of location:** describes the new location's contents and any associated updates.

  Action: goto restaurant
  Obs: You are now in the kitchen of your restaurant.
  You restock the fridge with raw lettuce.
  In the fridge, you see: raw tortillas, raw lettuce.
  In the cabinet, you see: olive oil; lime juice.
  There are 5 cooking tools in the environment:
  - pan_0, pan_1 are both empty. Capacity: 0/4.
  - cutting board, grill_0, grill_1 are all empty. Capacity: 0/1.
  You're not carrying anything right now.

- **Other actions:** the observation reflects the immediate outcome of that action.

  Action: put raw shrimp into pan_0
  Obs: You put raw shrimp into pan_0. Please wait for 3 timesteps for it to be cooked.

B.2.2.4 Action Space

The agent may choose one of the following actions at each step:

- `goto <location>`
- `harvest <crop>`
- `fish <seafood>`
- `buy <product>`
- `put <ingredient>into <tool>`: place ingredients into one of eight types of cooking tools.
- `chop <ingredient>`

- `wait`: wait for a timestep.
- `plate from <tool>into plate <id>`
- `serve plate<id>`

The current action space can be further decomposed into lower-level actions. For example, the `put` action could be separated into two steps: first `take` the ingredient, then `put` it into the specified tool.

Not all actions are guaranteed to succeed. For example, the environment includes multiple fishing harbors, but only unoccupied harbors are accessible. If the agent attempts to go to a harbor that is currently in use, the `goto` action will fail, and the observation will indicate that the location is unavailable. Similarly, the `fish` action has probabilistic outcomes. The agent may fail to catch anything or may catch a different type of seafood than intended. These outcomes encourage agents to reason under uncertainty.

B.2.2.5 Reward

The episode ends either when the agent serves all required dishes or the maximum number of steps is reached. The task is considered successful if all required dishes are correctly served. At the end of the episode, the agent receives a scalar reward based on the number of correctly completed dishes:

$$\text{reward} = \frac{\#\text{ of qualified dishes served}}{\#\text{ of required dishes}} \tag{B.2}$$

A dish is considered correct only if it exactly matches the corresponding recipe, as determined by the environment's internal validator. To be accepted, it must meet the following conditions:

- All required ingredients are present, and no incorrect or extra ingredients are included.
- Each ingredient must have the correct preparation state (e.g., `chopped and boiled`).
- All required seasonings must be applied, with correct timing and no extra seasonings added.

B.2.2.6 Guidance Generation

Perfect Guidance The agent receives complete guidance generated based on the dish's recipe template and the distribution of ingredients across locations, including the following components:

- Recipe Overview: a textual description of the required ingredients, seasonings, and cooking procedure, formatted in a natural and structured manner. For example:

  `Recipe for 1 serving of taco (Smoky Lime)`
  Ingredients: raw chicken, raw lettuce, raw shrimps, raw tortillas.
  Seasonings (add during cooking): cumin, lime juice, olive oil, paprika.
  Steps:
  - Grill the raw tortillas; then chop the raw lettuce.
  - Pan-fry the raw shrimp and raw chicken.
  - Add olive oil, paprika, cumin, and lime juice to the pan.
  - Wait for all ingredients to finish cooking, then assemble them from the cutting board, grill, and pan onto a single plate.

- Ingredient Locations: where each raw ingredient and seasoning can be found. For example:

  `Collect ingredients from the following locations:`
  Farm: lettuce
  Harbor_2: shrimp
  Store: chicken, cumin, paprika

Imperfect Guidance For imperfections in the guidance itself, we focus on two types: *underspecified* and *incomplete* instructions. To introduce environmental dynamics, we design scenarios with varying complexity, ranging from simple disruptions such as missing crops, where crops may have been stolen or are not yet ripe, to more challenging situations like mistakes made by a rookie chef. This spectrum allows us to evaluate agent robustness across different levels of difficulty.

1. Missing Ingredient Locations The agent is provided with a complete recipe but receives no information about the locations of the required ingredients. Unlike in the perfect guidance setting, this forces the agent to actively explore the environment to find each item.

2. Abstracted Steps Fine-grained operations and ingredient states in the recipe are replaced with high-level descriptions. For example, a detailed instruction such as *chop raw mushroom... boil the raw, chopped mushroom* may be simplified to *chop the mushroom... boil the mushroom*, omitting explicit references to intermediate states.

3. Similar Recipe Substitution Instead of receiving the exact recipe for the target dish, the agent is given a recipe for a similar dish. Key information such as ingredients, seasonings, and preparation states is still included, but step-by-step instructions are missing. This setup mimics retrieval-based guidance systems that return approximate but not fully aligned instructions.

4. Crop Disappearance Certain ingredients expected to be available at the farm may be missing due to simulated environmental events, such as animal interference or weather-related damage. To ensure the task remains solvable, the missing item is guaranteed to be stocked at an alternative location. The agent must detect the absence, adapt its plan, and navigate to the alternative source. This scenario provides a simple but essential test of the agent's ability to adapt when expected resources are unavailable.

5. Storage Loss Some ingredients that were initially available in the kitchen may later become unavailable due to being used up or spoiled by the time the agent returns from ingredient collection. Missing or depleted items can be discovered either upon arrival or when the agent attempts to use them, while spoiled ingredients are only revealed at the moment of use.

6. Novice Execution Errors In some episodes, the agent may follow the guidance correctly but still face problems due to simulated action noise, mimicking the kinds of mistakes a novice chef might make. For example, it may accidentally overseason a dish or apply an unintended transformation. The agent is expected to recognize the failure and adjust its behavior accordingly. This scenario evaluates the agent's ability to detect execution errors and maintain awareness of its own actions.

7. Tool Limitations The environment provides only a limited number of cooking tools (such as two pans or cutting boards), yet the agent may be required to prepare multiple servings. Since tools cannot be duplicated, the agent must learn to schedule actions efficiently, reusing tools across batches while also coordinating the timing of operations to complete the task under resource constraints. Rigidly following a one-serving guidance can lead to inefficient execution, as it may overlook opportunities for tool reuse, parallelism, and multi-serving coordination.

B.2.3 MIRAGE-World Gaming

B.2.3.1 Task Overview

In this domain, the agent plays a turn-based roguelike RPG with the goal of defeating the final-floor boss. The environment features procedurally generated levels, each belonging to one of four types: *growth* (item collection and crafting), *combat* (enemy encounters), *shop* (purchasing opportunities), and *boss* (final or intermediate mini-boss battles). Each level type offers distinct affordances, and success requires careful management of health, inventory capacity, and elemental effectiveness.

B.2.3.2 Task Generation

Each roguelike episode is procedurally generated based on a predefined level layout. In the standard configuration used throughout this chapter, the level sequence is fixed as:

$$\texttt{growth} \rightarrow \texttt{growth} \rightarrow \texttt{combat} \rightarrow \texttt{growth} \rightarrow \texttt{shop} \rightarrow \texttt{boss}$$

The framework also supports more complex variants by extending or modifying the sequence, such as adding extra levels or inserting intermediate challenges like a `miniboss`. Task generation begins by sampling a final boss, which is assigned one of eight possible elemental types: `FIRE`, `WATER`, `NATURE`, `LIGHT`, `DARK`, `ELECTRIC`, `ICE`, or `EARTH`. These types follow a predefined effectiveness matrix—for example, `ELECTRIC` is highly effective against `WATER` with a multiplier of 2.0 but ineffective against `EARTH` with a multiplier of 0.0. Based on the boss's element, the system identifies the most effective counter-element and samples a corresponding advanced weapon as the agent's ultimate goal. The components required to craft this weapon are then distributed across the levels using a backtracking planner, which ensures that the total number of items remains within the agent's inventory capacity and that distractor items are included to maintain realism and increase challenge. Item distribution is tailored to level types: in `growth` levels, items appear on the ground or inside containers; `combat` levels contain enemies that may drop items; and the `shop` level offers purchasable items. The agent's initial health is adjusted to match the expected maximum possible damage following an optimal plan, which assumes efficient crafting and equipping of the best available weapons for each enemy or miniboss encounter. A small safety margin is added to tolerate minor mistakes.

B.2.3.3 Observation Space

At each timestep, the agent receives a textual observation that reflects the outcome of its previous action and the current environmental context. Observations are composed of three main components, with the exact content determined by the type of action taken:

- **Action Feedback:** The result of the most recent action. The content varies based on the action type. For example, combat actions return damage exchanges such as: *You attack with Dark Dagger and deal 15 damage. Then, Ancient Flame Sprite attacks and deals 7 damage.*
- **Level Context:** A description of the agent's current surroundings, typically shown when entering a new level. The details depend on the level type:
  - `Growth Level:` Lists nearby *Collectible Items*, *Containers*, *Readable Objects*, and *NPCs*.
  - `Combat Level:` Lists current *Enemies*.
  - `Shop Level:` Displays available items for purchase, their costs, and optional NPC dialogue.
  - `Boss/Miniboss Level:` Introduces powerful enemies.
- **Character Sheet:** Shows the agent's internal state, updated as needed depending on the action. It includes HP, coin count, and current inventory (with elemental tags and item weights).

B.2.3.4 Action Space

The agent can issue one action per timestep, chosen from the following list:

- `collect <item>`: pick up a listed collectible item.
- `open <container>`: open a container and automatically collect its contents.
- `craft <item>`: combine the exact required materials in the inventory to produce a crafted item.
- `discard <item>`: drop an item to free up inventory capacity.
- `unequip`: unequip the currently held weapon.
- `equip <weapon>`: equip a weapon from the inventory.

- `attack <enemy>`: attack a target enemy.
- `buy <item>`: purchase a listed item from the shop.
- `talk <npc>`: initiate dialogue with a non-player character.
- `read <object>`: read a readable object (e.g., books, journals).
- `leave`: exit the current level and advance to the next.
- `back`: return to the previous level, if allowed.
- `recipes`: display the list of all available crafting recipes.

B.2.3.5 Reward

An episode ends when the agent defeats the boss, exceeds the step limit, or is defeated by enemies. The task is considered successful if the agent defeats the boss and finishes with positive HP. The reward is defined as the ratio of final HP to initial HP:

$$\text{reward} = \frac{\text{Final HP}}{\text{Initial HP}} \tag{B.3}$$

This encourages efficient combat and strategic planning. The agent must balance exploration, crafting, and combat to preserve health for the final encounter.

B.2.3.6 Guidance Generation

Perfect Guidance Given full control over level layout, item placement, and crafting dependencies, we generate an optimal path to task completion using a backtracking planner. This path is then translated into natural language guidance that help the agent defeat the final boss, which is automatically generated and reflect a valid, solvable strategy.

- **Action-oriented guidance**
  It provides details goals for each level, specifying how the agent should maintain. Example:

  > `Level 2 (growth):`
  > - Fire Essence ×1 (available in the tide pool or the wooden crate; only one is needed)
  > - Water Crystal ×1 (on the ground; used later for trading in the shop)

Imperfect Guidance We introduce several types of imperfect guidance. These modifications are designed to introduce uncertainty while preserving task solvability, either by allowing recovery or by embedding helpful clues in the environment.

1. Partial Removal In this setting, only a subset of the level-by-level plan is provided. For example, the guidance may include detailed instructions for Level 2 and Level 5, but omit information about Levels 3 or 4. This forces the agent to interpolate or plan autonomously for the missing levels.

2. Abstract Material List This variant provides a minimal form of guidance, without providing full information on how or where to acquire required materials. We implement two subtypes:

- **Global checklist.**
  A flat list of all required materials for the entire episode, presented without any per-level breakdown. The agent must determine when and where to obtain each item across the levels. Example:

  `You will need to prepare the following materials:`
  - Fire Essence ×3
  - Weapon Prototype ×1
  - Magic Catalyst ×1
  - Enchanted Cloth ×1
  - Water Crystal ×1

- **Per-level checklist.**
  A breakdown of required materials by level, without revealing where they appear or how to collect them. As a result, the agent must explore each level to locate items. Example:

  `Level 1 (growth):`
  - Fire Essence ×1

  `Level 2 (growth):`
  - Fire Essence ×1

  ...

3. Incorrect Crafting Recipe Here, the agent is provided with a slightly incorrect crafting recipe, for example, a recipe for a Fire Staff that omits one required component

or includes an invalid substitute. Attempting to craft with this recipe would result in a wrong item or trigger an error message. However, the environment supports a `recipes` function that the agent can invoke at any time to retrieve the correct recipes. This setting tests the agent's ability to recognize guidance failure, consult internal resources, and adjust accordingly.

4. Renamed Elements Some collectible items in the environment are renamed to unfamiliar terms. For example, `Fire Essence` may appear in guidance, while the corresponding item in the environment is labeled `Fire Residue`. To help the agent resolve this mismatch, we embed hints into the environment: NPCs may explain local terminology differences, or readable objects may contain clues about alternate names. Although the guidance-to-environment mapping is distorted, sufficient signals exist to bridge the vocabulary gap. This setup evaluates the agents ability to integrate environmental cues with imperfect guidance.

5. Localized Content Shuffling. In this variant, key items are shuffled within a level. For instance, instead of finding a Magic Catalyst on the ground, it may now appear inside a nearby chest, or a required item may drop from a different enemy than originally indicated. Importantly, the shuffling is restricted to within-level scope, preserving solvability. However, it challenges agents that rely too rigidly on detailed location-specific instructions, for example, expecting to find an item specifically in the "stone chest".

B.2.4 MIRAGE-Web WebArena

B.2.4.1 Overview

WebArena [188] is a high-fidelity, standalone web interaction environment designed to evaluate autonomous agents ability to execute real-world tasks via a simulated browser. It includes organically sourced content and fully functional web applications spanning diverse domains. In our benchmark, we focus on four representative task types: E-commerce website (OneStopShop), GitLab, Reddit, and an online store content management system (CMS). Each task is specified by a natural language objective along with high-level instructions, which may vary in imperfect guidance conditions. Our use of WebArena is consistent with its intended research purpose and license terms, extending it for robustness evaluation under imperfect guidance, maintaining its non-commercial, research-only use.

B.2.4.2 Observation Space

At each timestep, the agent receives an observation that mimics the human web browsing experience. The observation includes:

- **Task Intent**: The natural language description of the user's objective.
- **Current URL**: The URL of the web page currently being viewed.
- **Open Tabs**: A list of all currently opened tabs and their indices.
- **Focused Tab Content**: The content of the current web page, rendered as an accessibility tree.
- **Previous Actions**: A list of past actions taken by the agent.
- **Notes**: A list of notes the action took from the start of the trajectory.

B.2.4.3 Action Space

The agent can issue one action per step from a fixed set of browser-interaction primitives:

- **Page Operation Actions:**
  - `click [id]`: Clicks on an element with the specified identifier.
  - `type [id] [content] [press_enter_after=0|1]`: Types content into an input field.
  - `hover [id]`: Moves the cursor over the specified element.
  - `press [key_comb]`: Simulates pressing a keyboard shortcut (e.g., Ctrl+C).
  - `scroll [direction=up|down]`: Scrolls the page in the given direction.
- **Tab Management Actions:**
  - `new_tab`: Opens a new tab.
  - `tab_focus [tab_index]`: Switches focus to the specified tab.
  - `close_tab`: Closes the currently focused tab.
- **Navigation Actions:**
  - `goto [url]`: Navigates to the given URL.
  - `go_back`, `go_forward`: Moves backward or forward in browsing history.
- **Memory and Timing Actions:**
  - `write_to_note [content]`: Stores useful information as an internal note for later use.

* `wait`: Takes no interaction, simulating a wait action for dynamic content to load.

– **Completion:**

* `stop [answer]`: Ends the episode and optionally submits a text answer.

B.2.4.4 Guidance Generation

Perfect Guidance In MIRAGE-Web, perfect guidance is constructed by collecting high-quality human demonstrations and converting them into structured step-by-step action sequences. Specifically, we gather 450 successful human-completed trajectories across the four domains used in our benchmark: GitLab, Reddit, E-commerce Shopping, and Shopping-Admin(CMS). Human annotators followed the task descriptions and prompts already included in the WebArena framework, which specify the goal, observations, past actions, and action formats. The human annotations were performed by the authors. No external annotators were recruited or compensated. No demographic or geographic information is applicable as well. Each trajectory is then translated into a sequence of natural language instructions that align with the action space.

To improve clarity, we augment raw action traces with contextual information extracted from the accessibility tree when elements appear multiple times on the same page. For example, multiple links labeled "Edit" or "Issues" may exist; in such cases, we append brief references to nearby elements or structural cues to help the agent identify the correct target. These descriptions are concise but informative, enabling robust interpretation without requiring pixel-level inputs. An example excerpt of a perfect guidance sequence is shown below:

1. type [element_id] [primer design] where [element_id] is searchbox Filter by name
2. click [element_id] where [element_id] is heading Primer / design
3. click [element_id] where [element_id] is link Issues; There are multiple Issues on the page, this specific one is between link Project information and StaticText 21
4. click [element_id] where [element_id] is link Edit; There are multiple Edit links, this one is between StaticText Assignee and link Inayaili León
5. stop [your_answer]

Imperfect Guidance We implement several forms of imperfect guidance to evaluate agent robustness in realistic web navigation scenarios. Some variants are generated programmatically, while others are produced by prompting a language model to rewrite the

original guidance. These LLM-generated rewrites sometimes contain noise or hallucinations, reflecting the kinds of errors commonly seen in LLM-generated guidance, tips, or skills. To ensure quality and consistency, we also create a cleaned version of each variant by having human annotators read and rewrite the steps manually. Below, we describe the different types of imperfect guidance used in our benchmark:

1. Skip Step In this variant, we randomly drop one step from the original action sequence. This simulates a common human oversight where a user forgets to specify an intermediate action, such as failing to mention that a menu needs to be opened before a link becomes visible. The agent must detect that something is missing and recover by reasoning or trial and error.

2. Narrative Rewrite We prompt a language model to translate each step of the action sequence into a natural language command. For example, the original instruction "*click [element_id] where [element_id] is link Orders*" may be rewritten as "*Click on the Orders link*". This format resembles how users typically express instructions in a more casual and descriptive manner, which may introduce ambiguity or underspecification.

3. Merged Steps This variant goes further than the narrative rewrite by combining multiple low-level actions into a single sentence. More specifically, we prompt LLMs to merge action steps that are on the same webpage. For example:

Original: `scroll [down]`
`scroll [down]`
`scroll [down]`
`scroll [down]`
Merged: `Scroll down four times.`

Such merged forms reflect how users might batch repeated or related actions into higher-level abstractions. The agent must learn to map this concise guidance back into multiple concrete steps.

4. UI Change Rather than modifying the web environment directly, we simulate UI-level variability by altering the element descriptors in the guidance. For example, the original step "*click [element_id] where [element_id] is link* ***Orders***" may be rewritten as "*click [element_id] where [element_id] is link* ***All Orders***". These changes involve

replacing text with near-synonyms, abbreviations, or slightly altered phrasing, rather than introducing entirely incorrect meanings.

B.2.5 Baselines

5. Observation Delay In this condition, the agent's action is successful, but the updated observation is delayed. This reflects real-world phenomena such as slow internet connections or UI latency. The challenge is that the agent must not assume failure, nor proceed too quickly based on outdated state.

B.3 Evaluation Setting

In this section, we will describe the experimental settings, models used, compute resources, and results with different types of imperfect guidance.

We evaluate three agent baselines in the MIRAGE-World setting: ReAct, Plan-and-Solve (P&S), and Reflexion. These methods represent different approaches to reasoning and action planning under imperfect guidance. Below, we briefly describe how each method is adapted and implemented.

ReAct We follow the ReAct framework [170], which interleaves natural language reasoning with environment actions. To support flexible reasoning, we extend the action space by introducing a `think` action. This allows the agent to generate thoughts that do not affect the environment and only return an observation "OK". The agent is prompted in a one-shot format, with a demonstration showing how to interleave reasoning and actions throughout the episode.

Plan-and-Solve (P&S) The P&S baseline uses a hierarchical planning structure inspired by prior work [153, 56]. A high-level planner is prompted with a one-shot demonstration and takes as input the full episodic history, previously proposed high-level steps, and any provided guidance. It is tasked with generating the next high-level plan that can be achieved in a few actions. A separate ReAct-style agent is then prompted to execute this step. The sub-process ends when the agent issues a `stop` action or exceeds a predefined step budget (twelve steps for navigation and cooking, fifteen for gaming).

Env	Guidance	G.	GPT-4.1 mini	GPT-4o mini	Qwen3 14B	LLaMA3 70B	Qwen3 235B
Navigation							
Nav	tile-based	P	100.0 ± 0.0	94.0 ± 0.0	99.0 ± 1.0	90.0 ± 3.0	97.0 ± 2.1
	turn-based	P	64.0 ± 2.4	29.0 ± 2.8	30.0 ± 2.6	10.0 ± 2.1	27.0 ± 5.8
	obs-based	P	90.0 ± 3.2	60.0 ± 4.3	44.0 ± 3.4	15.0 ± 3.1	50.0 ± 5.8
	wrong tiles	IN	68.0 ± 3.7	38.0 ± 5.1	20.0 ± 3.0	13.0 ± 4.2	34.0 ± 6.2
	wrong turns	IN	76.0 ± 4.0	23.0 ± 4.2	27.0 ± 4.2	10.0 ± 3.8	27.0 ± 4.0
	None	NA	30.0 ± 8.6	32.0 ± 4.9	10.0 ± 2.6	1.0 ± 1.0	32.0 ± 5.7
Nav-Season	obs-based	M	90.0 ± 3.2	52.0 ± 12.0	43.0 ± 6.3	17.0 ± 3.7	56.0 ± 3.7
Nav-Block	tile-based	M	82.0 ± 6.6	16.0 ± 8.7	6.0 ± 2.7	5.0 ± 2.7	10.0 ± 3.3
	turn-based	M	72.0 ± 7.3	16.0 ± 3.7	18.0 ± 2.9	9.0 ± 3.1	19.0 ± 4.1
	obs-based	M	78.0 ± 7.3	6.0 ± 4.0	16.0 ± 4.3	6.0 ± 2.2	16.0 ± 4.3
Nav-Dynamic	tile-based	M	80.0 ± 4.5	70.0 ± 4.5	81.0 ± 3.5	70.0 ± 3.3	75.0 ± 4.3
	turn-based	M	46.0 ± 9.3	7.0 ± 2.6	15.0 ± 5.8	6.0 ± 2.2	25.0 ± 2.2
	obs-based	M	74.0 ± 6.8	26.0 ± 6.0	45.0 ± 5.2	8.0 ± 2.5	49.0 ± 7.2
Cooking							
Cook	full info	P	98.0 ± 2.0	66.0 ± 6.8	88.0 ± 5.0	78.0 ± 2.9	93.0 ± 2.1
	no ing location	IC	96.0 ± 2.4	48.0 ± 9.7	88.0 ± 2.9	76.0 ± 2.7	94.0 ± 2.2
	abstracted steps	U	94.0 ± 2.4	48.0 ± 8.6	79.0 ± 3.5	77.0 ± 4.2	90.0 ± 3.0
	similar recipe	M	88.0 ± 2.0	30.0 ± 7.1	46.0 ± 3.4	56.0 ± 4.8	76.0 ± 3.7
Cook-Crops	full info	M	92.0 ± 4.9	42.0 ± 7.3	46.0 ± 5.2	42.0 ± 6.3	60.0 ± 3.0
Cook-Storage	full info	M	96.0 ± 2.4	60.0 ± 7.1	52.0 ± 3.6	43.0 ± 3.7	70.0 ± 3.0
Cook-Rookie	full info	M	88.0 ± 5.8	40.0 ± 8.9	43.0 ± 2.6	21.0 ± 4.1	81.0 ± 4.8
Roguelike							
Rogue	action-oriented	P	70.0 ± 18.2	46.0 ± 7.6	45.0 ± 6.2	44.0 ± 4.0	69.0 ± 4.8
	global checklist	U	42.0 ± 14.6	16.0 ± 3.6	4.0 ± 1.6	0.0 ± 0.0	29.0 ± 6.0
	per-level checklist	U	62.0 ± 16.9	28.0 ± 4.1	27.0 ± 4.7	4.0 ± 2.2	49.0 ± 5.3
	partial removal	IC	48.0 ± 13.9	28.0 ± 4.7	17.0 ± 3.4	11.0 ± 3.1	32.0 ± 3.9
	incorrect recipe	IN	12.0 ± 5.8	16.0 ± 2.8	19.0 ± 2.3	22.0 ± 4.9	26.0 ± 6.2
	None	NA	16.0 ± 8.1	10.0 ± 3.9	1.0 ± 1.0	0.0 ± 0.0	8.0 ± 4.2
Rogue-shuffle	action-oriented	M	50.0 ± 20.5	16.0 ± 5.7	37.0 ± 4.0	18.0 ± 3.9	38.0 ± 6.3
Rogue-rename	action-oriented	M	34.0 ± 14.0	14.0 ± 2.8	20.0 ± 3.7	26.0 ± 4.5	45.0 ± 6.5

Table B.1: MIRAGE-World results for ReAct across different guidance conditions. All results are from a single trial per task. Each number denotes the success rate (%) out of 100 tasks (10 tasks across 10 seeds), rounded to the nearest integer. Standard errors are computed across seeds. The column labeled **G.** indicates the type of guidance: **P** = Perfect, **NA** = No guidance, **IN** = Incorrect, **IC** = Incomplete, **M** = Misaligned, **U** = Underspecified.

Reflexion We implement Reflexion [132] to examine whether self-reflection can improve agent performance across trials. Unlike the other baselines, Reflexion is allowed up to three full attempts per task. After each trial, the agent is shown a brief summary of general, high-level possible failure points and is prompted to reflect in an open-ended way. The environment is then rewound to its initial state, and the agent begins a new trial. Its prompt is augmented with a list of reflections and suggestions for the new trial from all previous attempts. Although Reflexion is not directly comparable to single-episode agents, it demonstrates how iterative reasoning and feedback can improve performance and help reveal task solvability.

For MIRAGE-Web, we use a ReAct-style agent. The agent is prompted with a two-shot demonstration and must think through the task step by step before deciding on an action. The final action must be wrapped in a code block enclosed by triple backticks, following the required action format.

B.3.1 Hyper-parameters

For all MIRAGE-World experiments, we use the following language models: GPT-4.1-mini [70], GPT-4o-mini [94], Llama-3-70B [39], Qwen-3-14B [118], and Qwen-3-235B [118]. All model outputs are generated with temperature set to zero to ensure deterministic behavior, allowing us to isolate the impact of guidance variation without stochasticity from decoding. Each type of imperfection is evaluated with 10 random seeds, each consisting of 10 unique task instances, resulting in 100 total tasks per condition.

For MIRAGE-Web, we use GPT-4o [101], as the tasks involve more realistic and challenging web tasks with long, complex observations that demand stronger contextual reasoning.

B.3.2 Compute Resources

We utilized the OpenAI API to run inference for all OpenAI models tested, including GPT-4o, GPT-4.1-mini, and GPT-4o-mini. The wall-clock runtime varies across tasks and models. For example, for navigation tasks, GPT-4o-mini takes about 12 minutes per 10 tasks, while GPT-4.1-mini completes the same batch in approximately 11 minutes. On cooking tasks, GPT-4.1-mini takes around 8 minutes, and for gaming, about 12.5 minutes for 10 sequential tasks. Running a full set of 450 tasks for a single imperfection type in MIRAGE-Web using GPT-4o takes approximately 10 hours when executed sequentially.

Running the whole MIRAGE-World with gpt-4o-mini for 100 tasks each imperfection type will cost about 70 US dollars, and running the whole MIRAGE-Web with gpt-4o costs about 300 US dollars.

We utilized an internal computing cluster to run inference on all open-weight models tested (Qwen-3 14B, Qwen-3 235B, LLaMA-3 70B, etc.). This computing cluster consists of five machines with the following specifications:

1. 8 x RTX A6000 Ada (48 GB VRAM), Xeon E5-2697A v4 (2.60GHz), 1024 GB RAM
2. 6 x RTX A6000 (48 GB VRAM), AMD EPYC 7313 (3.00GHz), 1024 GB RAM
3. 8 x RTX A5000 (24 GB VRAM), Xeon E5-2630 v4 (2.20GHz), 256 GB RAM
4. 8 x RTX 4070 (12 GB VRAM), Xeon E5-2697 v4 (2.30GHz), 256 GB RAM
5. 8 x TITAN X (12 GB VRAM), Xeon E5-2630 v3 (2.40GHz), 256 GB RAM

We utilized a combination of Ollama and vLLM to deploy and run inference on open-source models. The average runtime varies across domains and task types. For the largest model, Qwen-3 235B, running 10 tasks took approximately 26 minutes for navigation, 20 minutes for cooking, and 43 minutes for gaming.

B.3.3 Results

B.3.3.1 MIRAGE-World Results

We present the full results of ReAct-style agents across different models in Table B.1. For all experiments, we report the success rate over 10 tasks for each of 10 seeds, and include standard error across seeds.

We additionally tested smaller models under the ReAct setting on MIRAGE-World navigation tasks. The results are shown in Table B.2. However, these models perform poorly on the benchmark, potentially due to the challenging nature of the tasks, which involves long-horizon reasoning and rich contextual understanding.

B.3.3.2 MIRAGE-Web Results

Results for MIRAGE-Web are shown in Table B.4, evaluated using GPT-4o with a ReAct-style agent across different types of guidance imperfections. We consider the following variants:

Model	None	M	I	Perfect
LLaMA3.2-3B	0%	0%	0%	0%
Qwen2.5-3B	0%	2%	0%	5%
Qwen2.5-7B	2%	9%	8%	14%

Table B.2: ReAct setting: success rates (%) under different guidance qualities for smaller models. M = Misaligned, I = Incorrect.

- **Skip Step:** randomly removes one step from the original guidance sequence.
- **Narrative:** reformulates the original structured actions into free-form natural language.
- **Merged:** combines steps that occur on the same webpage (i.e., same URL) and expresses them as a single verbal instruction.
- **UI Change:** apply LLMs to rewrites up to three UI elements in the observation using semantically similar alternatives.
- **Observation Delay:** introduces a delay of 2 to 5 timesteps before the agent receives the resulting observation.

Each imperfection simulates realistic challenges in interpreting and following web-based instructions, and allows us to assess the robustness of agents under varied forms of guidance degradation.

B.3.4 Experiments with Task-level Ambiguity

MIRAGE focuses on imperfect guidance under a well-defined goal (task). Some other interesting real-world scenarios include task ambiguity. While the aspect is orthogonal to our focus, it can be combined with MIRAGE tasks. Our *underspecified* guidance category already captures part of the space. We additionally implement a cooking task with ambiguous goals (e.g., multiple sushi recipes while the customer only says "I want sushi").

We then augment the action space with `ask[...]` to query a simulated customer (powered by `gpt-4o-mini` with access to the task spec). We evaluate: (i) Standard ReAct with a demonstration of an unambiguous task; and (ii) Imperfect-aware ReAct with a demonstration showing inquiry to resolve ambiguity. Table B.5 shows that ambiguity harms ReAct, but is easy to identify and recover via imperfect-aware prompting.

Env	Guidance	G.	ReAct	Plan & Solve	Reflexion
Navigation					
Nav	tile-based	P	100.0±0.0	78.0 ± 5.7	100.0±0.0
	turn-based	P	64.0 ± 2.4	20.0 ± 4.1	68.0 ± 3.7
	obs-based	P	90.0 ± 3.2	69.0 ± 9.4	90.0 ± 3.2
	wrong tiles	IN	68.0 ± 3.7	29.0 ± 3.0	70.0 ± 3.2
	wrong turns	IN	76.0 ± 4.0	21.0 ± 3.0	80.0 ± 3.2
	None	NA	30.0 ± 8.6	10.0 ± 4.4	30.0 ± 8.6
Nav-Season	obs-based	M	90.0 ± 3.2	47.0 ± 4.1	92.0 ± 2.0
Nav-Block	tile-based	M	82.0 ± 6.6	5.0 ± 1.7	84.0 ± 5.1
	turn-based	M	72.0 ± 7.3	25.0 ± 2.3	78.0 ± 6.6
	obs-based	M	78.0 ± 7.3	0.0 ± 0.0	80.0 ± 7.7
Nav-Dynamic	tile-based	M	80.0 ± 4.5	28.0 ± 2.1	80.0 ± 4.5
	turn-based	M	46.0 ± 9.3	22.0 ± 2.5	48.0±10.7
	obs-based	M	74.0 ± 6.8	25.0 ± 2.7	76.0 ± 5.1
Cooking					
Cook	full info	P	98.0 ± 2.0	72.0 ± 6.2	98.0 ± 2.0
	no ingredient location	IC	96.0 ± 2.4	50.0 ± 6.5	98.0 ± 2.0
	abstracted steps	U	94.0 ± 2.4	60.0 ± 5.2	94.0 ± 2.4
	similar recipe	M	88.0 ± 2.0	63.0 ± 5.7	94.0 ± 2.4
Cook-Crops	full info	M	92.0 ± 4.9	48.0 ± 4.8	92.0 ± 4.9
Cook-Storage	full info	M	96.0 ± 2.4	41.0 ± 4.2	96.0 ± 2.4
Cook-Rookie	full info	M	88.0 ± 5.8	38.0 ± 6.6	92.0 ± 4.9
Roguelike					
Rogue	action-oriented	P	70.0±18.2	56.0 ± 2.8	70.0±18.2
	global checklist	U	42.0±14.6	23.0 ± 4.8	46.0±14.4
	per-level checklist	U	62.0±16.9	40.0 ± 6.2	64.0±17.2
	partial removal	IC	48.0±13.9	42.0 ± 5.7	50.0±13.8
	incorrect recipe	IN	12.0 ± 5.8	10.0 ± 2.2	12.0 ± 5.8
	None	NA	16.0 ± 8.1	8.0 ± 1.7	18.0 ± 8.0
Rogue-shuffle	action-oriented	M	50.0±20.5	18.0 ± 3.5	54.0±22.3
Rogue-rename	action-oriented	M	34.0±14.0	19.0 ± 5.8	34.0±14.0

Table B.3: Comparison of ReAct, Plan & Solve, and Reflexion using GPT-4.1-mini across all tasks. Numbers denote success rates (%) averaged over 10 tasks per seed, with 10 different seeds. Reflexion allows 3 trials per task, while the others use only 1. Standard errors are computed across seeds.

Domain	Perfect	Incomplete	Underspecified		Incorrect	Misaligned	None
		Skip Step	Narrative	Merged	UI Change	Obs Delay	
Reddit	86.67% ± 3.32%	37.14% ± 4.72%	35.23% ± 4.67%	40.95% ± 4.80%	56.19% ± 4.84%	35.24% ± 4.66%	11.43% ± 3.23%
Shopping	62.14% ± 4.10%	38.57% ± 4.11%	44.29% ± 4.20%	42.86% ± 4.18%	53.57% ± 4.21%	37.14% ± 4.08%	32.09% ± 3.90%
CMS	43.75% ± 4.69%	18.75% ± 3.69%	24.11% ± 4.04%	18.75% ± 3.69%	35.71% ± 4.53%	18.75% ± 3.69%	15.18% ± 3.40%
GitLab	57.29% ± 5.05%	36.46% ± 4.91%	38.54% ± 4.97%	35.42% ± 4.88%	40.63% ± 5.01%	30.21% ± 4.69%	26.04% ± 4.48%

Table B.4: Task success rate (%) and standard error for ReAct (GPT-4o) on WebArena tasks under different guidance conditions. Columns are ordered left to right from perfect guidance to imperfect guidance, and then to no guidance.

Task Type	Std-R	Imp-R
Ambig. (new)	37%	54%
Unambig. (original)	64%	N/A

Table B.5: Performance on ambiguous vs. unambiguous cooking tasks with inquiry action. Abbreviations: **Std-R** = Standard ReAct; **Imp-R** = Imperfect-Aware ReAct; **Ambig. (new)** = ambiguous task with perfect guidance; **Unambig. (original)** = unambiguous task with perfect guidance.

B.4 Prompts

B.4.1 Imperfect Guidance Generation

We prompt GPT-4o to rewrite the original step-by-step guidance into several imperfect guidance formats. These rewrites are designed to reflect common issues in real-world guidance, such as ambiguity, abstraction, and UI mismatch. Figure B.3, Figure B.4, and Figure B.2 show representative examples of the three LLM-rewritten variants.

For the **narrative** and **UI change** variants, we process steps individually in order. In the UI change setting, we sample at most three steps to rewrite per task, focusing on those with clickable elements. The LLM is prompted to modify the element label using near-synonyms, abbreviations, or vague descriptions, while preserving overall intent. For the **merged** variant, we group consecutive steps that share the same URL and send them

together to the language model as a batch. This produces more abstract and compressed instructions that span multiple actions, mimicking how humans often summarize multi-step behaviors in narrative form.

To avoid potential hallucinations and mistakes, our submitted code includes a version of each rewritten variant that has been manually reviewed and refined by human annotators.

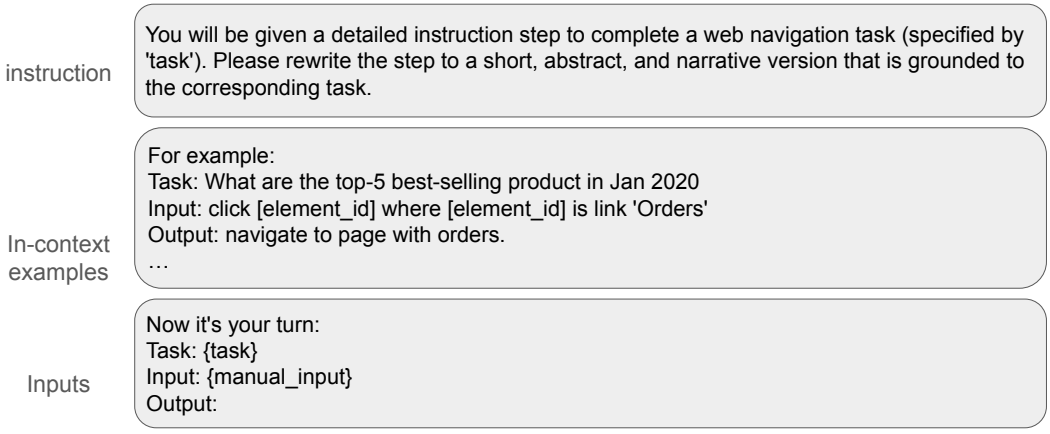


Figure B.2: Example of narrative-style rewrite. Guidance is rewritten in a more descriptive form.

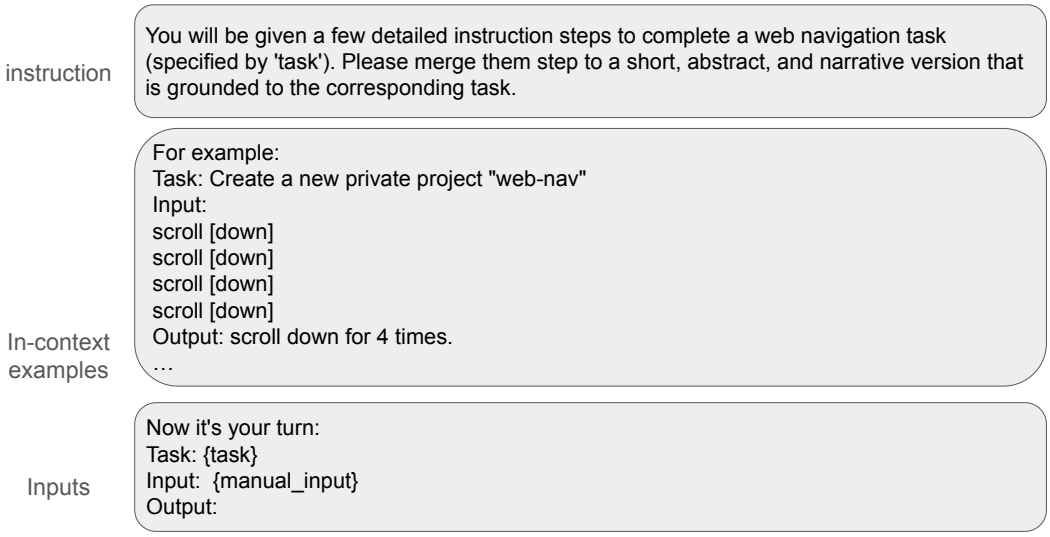


Figure B.3: Example of merged rewrite. Multiple consecutive steps are compressed into a single abstract description.

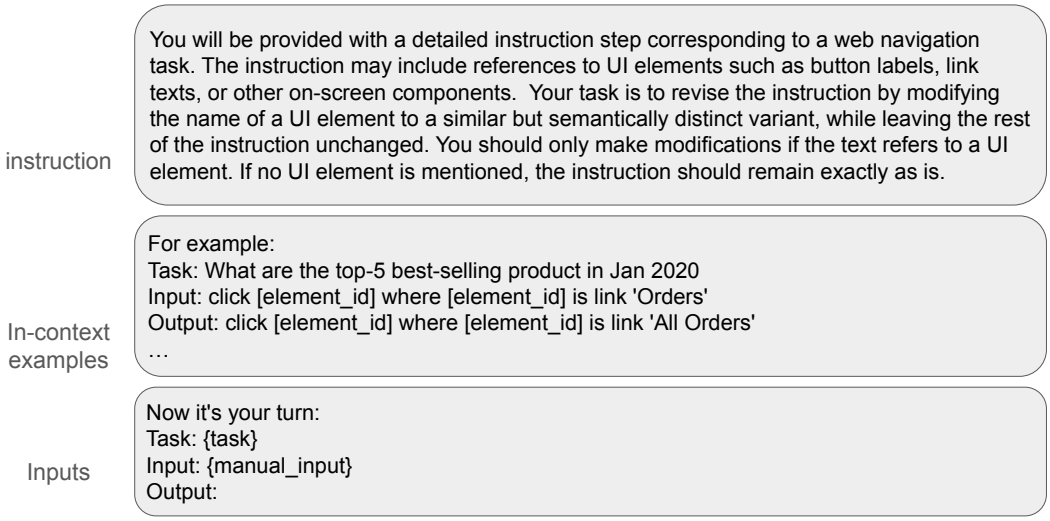


Figure B.4: Example of UI label variation. Labels for interface elements are changed to introduce ambiguity or misalignment.

APPENDIX C

Go Beyond Imagination: Maximizing Episodic Reachability with World Models

C.1 Implementation Details

C.1.1 Experiments on MiniGrid

Baselines Implementations of GoBI, NovelD [181], RIDE [120], RND [12], and EC [126] are built on the official codebase of NovelD. For fair comparisons, only the intrinsic reward r^{int} differs among the methods and they all use the same base algorithm IMPALA [27]. At the same time, all the experiments are run with the same compute resource with Nvidia TITAN X GPU and 40 CPUs. For NovelD, we rerun their official code to get the results of MiniGrid MultiRoom and KeyCorridor. For the experiments on ObstructedMaze, we did not find the proper hyper-parameters to fully reproduce their results. Therefore, we directly take the results reported in their paper. For RIDE and RND, we run the code in the official codebase of NovelD. For EC [126], their original paper does not include experiments on MiniGrid environments. Therefore, we implement our own version and tune the hyper-parameters with grid search. The intrinsic reward functions of GoBI and the baselines are listed below:

- GoBI: $(m_{t+1} - m_t)/\sqrt{N(o_{t+1})}$, where m_t is the size of the episodic buffer $\mathcal{M}$ and $N(o_{t+1})$ is the lifelong count of the observation o_{t+1} starting from the beginning of training.
- RND: $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$, which is the difference between a fixed random network $\hat{\phi}$ and a trained state embedding network ϕ. Here, ϕ is trained to minimize the same error.
- NovelD: $\max[novelty(o_{t+1}) - \alpha \cdot novelty(o_t), 0] * \mathbb{1}\{N^{epi}(s_{t+1} = 1)\}$. They apply RND to measure the novelty of o_t, i.e., $novelty(o_t) = \|\phi(o_t) - \hat{\phi}(o_t)\|^2$. $N^{epi}(s_{t+1} = 1)$

checks if the agent visits state s_{t+1} for the first time in an episode. Notice that they use the full environment information, i.e., everything in the grid world instead of only the 7×7 partially-observable view. Therefore N^{epi} counts s_{t+1} instead of o_{t+1}.

- RIDE: $\|\phi(o_t) - \phi(o_{t+1})\|^2 / \sqrt{N^{epi}(s_{t+1})}$, where ϕ is the state embedding network trained to minimize the prediction error of an inverse and a forward dynamics. N^{epi} indicates the episodic counts. Same as NovelD, in RIDE, they also use the state information s_{t+1} for episodic count.
- EC: $\beta - C(\mathcal{M}, o_{t+1})$, where $C(\mathcal{M}, o_{t+1})$ is the 90-th percentile similarity scores between o_{t+1} and all the observations in the episodic buffer $\mathcal{M}$. The similarity scores are calculated using a pre-trained episodic curiosity module. β is a hyper-parameter.

Policy and Value Function Training For fair comparisons, the policy network and value function network are the same for all approaches. The input observations of dimension $7 \times 7 \times 3$ are put into a shared feature extraction network, which includes three convolutional layers of kernel size= 3×3, padding= 1, channel=$32, 128, 512$, and stride= $1, 2, 2$ respectively with ELU activation. The features are then flattened and put through 2 linear layers with 1024 units and ReLU activation, and an LSTM layer with 1024 units. This shared feature is passed separately to 2 fully-connected layers with 1024 units to output action distribution and value estimation.

Dynamics Model For our implementation of the dynamics model, our input is the panorama of the current step. To get the panorama, we let the agent rotate for 3 times and concatenate the 4 observations to get inputs of size $28 \times 7 \times 3$. It is then passed to a feature extraction module that has the same structure as our policy and value function networks, except that the input to the first linear layer is 4×1024. We then concatenate it with actions and put it through a decoder with 2 linear layers of sizes 256 and 512, and reshape back to $7 \times 7 \times 3$ to get a predicted observation. We pre-train the dynamics model using 1e5 $(pano_t, a_t, o_{t+1})$ pairs collected by a random policy. RIDE also requires training dynamics models for the state embedding network ϕ. The input of their dynamics model is the state embedding and action. The forward model contains two fully-connected layers with 256 and 128 units activated by ReLU. The inverse dynamics model contains two fully-connected layers with 256 units and a ReLU activation function. Its input is the state embeddings of two consecutive steps.

Hash Functions We directly apply the default Python hashing function to hash the $7\times7\times3$ observations and predicted future observations before adding them to the episodic buffer.

State embedding NovelD, RIDE, and RND all require training a state embedding network ϕ. The input is the observation in MiniGrid with dimension $7\times7\times3$. It contains three convolutional layers with kernel size$= 3 \times 3$, padding $= 1$, stride $= 1, 2, 2$, number of channels $= 32, 128, 512$ respectively. The activation function is ELU. Following the convolutional layers are two linear layers of 2048 and 1024 units with ReLU activation.

Visitation Count For GoBI, $N(o)$ stores the flattened $7 \times 7 \times 3$ observations of each step. And for NovelD and RIDE, they count the full states at episodic level, whose shape varies from environment to environment. For example, the shape is $25 \times 25 \times 3$ for MultiRoom environments.

Hyper-parameters Table C.1 shows the values of hyper-parameters shared across different methods.

Parameter name	Value
Batch Size	32
Optimizer	RMSProp
Learning Rate	0.0001
LSTM Steps	100
Discount Factor γ	0.99
Weight of Policy Entropy Loss	0.0005
Weight of Value Function Loss	0.5

Table C.1: Hyper-parameters for experiments on MiniGrid. These hyper-parameters are shared across all the methods

For all of our experiments using GoBI on MiniGrid, we set the intrinsic reward coefficient $\lambda = 0.01$ and $k = 1$, which means only forwarding the dynamics model by 1 step. At the same time, as the action space is small and discrete, instead of randomly sampling some actions, we directly predict the future observations using all 7 possible actions. We list the hyper-parameter choices of intrinsic decay factor ρ in Table C.2. The value of ρ is chosen to make the intrinsic reward large at the beginning of training and near-zero at the end of the training.

Parameters	Value
Forward Step k	1
Intrinsic Decay ρ	6e−7 for MR-N7S8; 8e−7 for MR-N12S10, MR-N6; 1.5e−6 for KC-S3R3; 5e−7 for KC-S4R3, KC-S5R3 3e−7 for KC-S6R3, OM-2Dlh 2e−7 for OM-1Q, OM-2Dlhb, OM-2Q 5e−8 for OM-Full
$\hat{f}_\phi$ Optimizer	Adam
$\hat{f}_\phi$ Learning Rate	5e−4

Table C.2: The hyper-parameters of GoBI for experiments on MiniGrid.

For NovelD, we set $\lambda = 0.05$ for all the environments as suggested in their official codebase. For RIDE, we use $\lambda = 0.1$ on KeyCorridor-S3R3 and $\lambda = 0.5$ on all other environments. For RND, we set $\lambda = 0.1$ on all the environments. For EC, we make $\lambda = 0.01$ so that the initial average intrinsic reward of EC is similar to ours.

C.1.2 Experiments on DeepMind Control Suite

Baselines Implementations of GoBI, RE3 [130], ICM [108], and RND [12] are built on the official codebase of RE3. All the experiments apply the same base reinforcement learning algorithm RAD [74]. For RE3, we rerun their official code to get the results on all four environments. For ICM and RND, we follow the implementation details listed in RE3 to implement them to be compatible with DeepMind Control tasks. For a fair comparison, only the intrinsic reward design differs among the methods. The intrinsic reward functions of GoBI and the baselines are listed below:

- GoBI: $(m_{t+1} - m_t) \times \log(||y_i - y_i^{k-NN}||_2 + 1)$, where m_t is the size of the episodic buffer $\mathcal{M}$. The latter part is the RE3 intrinsic reward which we introduce below.
- RE3: $\log(||y_i - y_i^{k-NN}||_2 + 1)$, where $y_i = f_\theta(s_i)$ is a fixed representation outputs from a randomly initialized encoder and y_i^{k-NN} is a set of k-nearest neighbors of y_i among all the collected y's from the beginning of training.
- ICM: $\frac{\eta}{2}||\hat{\phi}(o_{t+1}) - \phi(o_{t+1})||_2^2$, where η is a scaling factor. $\phi(o)$ is a feature vector that is jointly optimized with a forward prediction model and an inverse dynamics model and $\hat{\phi}(o)$ predicts the feature encoding at timestep $t + 1$.

– RND: $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$, which is the difference between a fixed random network $\hat{\phi}$ and a trained state embedding network ϕ. Here, ϕ is trained to minimize $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$.

Architecture The observation size of all the environments is $84 \times 84 \times 3$. The encoder architecture follows the same one as in [172], which contains 4 convolutional layers of 3×3 kernels, channel=32, and stride=$2, 1, 1, 1$ with ReLU activations. The output is then passed to a fully-connected layer and normalized by LayerNorm.

Dynamics Model For the forward dynamics model that we use to generate future predictions, we apply the same world model structure as in Dreamer [46]. The input size of Dreamer is $64 \times 64 \times 3$. We down-sample the input observations to 64×64 instead of tuning the world model layers. We train the dynamics model together with the RL policy in an online manner instead of pre-train it because it takes many episodes for the predictions to be visually reasonable. Therefore it does not add extra effort to determine how much data should we collect to pre-train the dynamics model.

Image hashing As the observations are images in high-dimensional space and the predictions are usually not accurate, we hash the images to lower dimension to avoid taking too much space and to collapse similar observations and predictions. Following [146], we use the simple SimHash function to map the images to 50 bits. More specifically, we project the flattened images to a random initialized vector and use the signs of output vector values as the hash code.

Hyper-parameters Table C.3 shows the values of hyper-parameters shared across different methods.

For the intrinsic reward coefficients, we follow the best choices reported in RE3. For GoBI, we apply the same intrinsic reward coefficient λ and intrinsic reward decay ρ as the ones in RE3 for fair comparison. The intrinsic rewards that are specific to our method is shown in Table C.4. For the number of random actions n, we perform hyper-parameter search over {3, 5, 10, 20} and find that $n = 5$ perform well across all the tasks. For the number of forward steps k, we perform hyper-parameter search over {1, 2, 3, 5} and report the ones with the best results.

Parameter name	Value
Augmentation	Crop
Observation Size	(84, 84)
Action Repeat	2
Replay Buffer Size	100000
Initial Random Exploration Steps	1000
Frame Stack	3
Actor Learning Rate	0.0002
Critic Learning Rate	0.0002
Batch Size	512
# Nearest Neighbors	3
Critic Target Update Freq	2

Table C.3: Hyper-parameters for experiments on DeepMind Control Suite. These hyper-parameters are shared across all the methods

Parameter name	Value
# Forward Step k	3 for pendulum-swingup; 1 for others
# Random Actions n	5

Table C.4: Hyper-parameters for experiments on DeepMind Control Suite. These hyper-parameters are specific to our method.

C.2 Hyper-Parameters

C.2.1 Intrinsic Reward Decay

In Figure C.1, we show the training performance with different intrinsic reward decay ρ. The choice of ρ is to balance between the relative importance of the extrinsic and intrinsic reward. If ρ is too large, for example when $\rho = 1\text{e} - 6$, before the agent finds any goal, the intrinsic reward already decreases to very small. Therefore sometimes it is hard for the agent to learn anything useful, resulting in unsatisfactory performance. Meanwhile, if ρ is too small, for example when $\rho = 5\text{e} - 7$, the intrinsic reward will be too large at later stage of training and make the agent focus less on the extrinsic reward. Therefore the policy may converge slower.

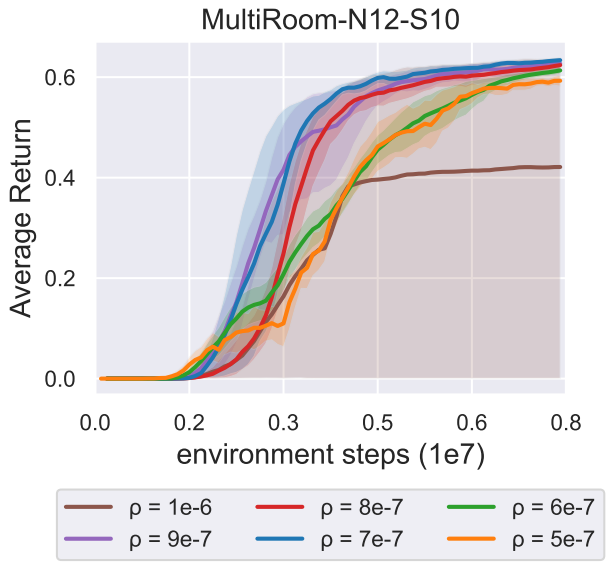


Figure C.1: Training performance of GoBI on MiniGrid MultiRoom-N12-S10 with different intrinsic reward decay ρ.

C.2.2 Number of Randomly Sampled Actions

In the MiniGrid experiments reported in Section 4.3, we do not randomly sample actions because MiniGrid has a small discrete action space with only 7 actions. Therefore we directly predict future observations of all 7 actions. However, we also report the results with $n = 3, 5, 10, 15$ random actions in Figure C.2. To summarize, $n = 5, 10, 15$ all have similar performance on MiniGrid, while a larger n makes the wall-clock training time longer. $n = 3$ is slightly slower at the early stage, but still outperforms the previous state-of-the-art methods. Overall, $n = 5$ would be a good choice for the MiniGrid environments.

C.3 Exploration Behaviour Comparison between GoBI and EC

In Figure C.3, we visualize the policy visitation heatmaps of GoBI and EC [126] on a MultiRoom environment from MiniGrid. Although EC fails to learn an optimal policy, we can still capture its preference from the heatmaps. An agent trained with EC prefers going to the corners of the room, which generally have lower similarity scores than the states in the middle of the room. However, if the similarity scores are not low enough

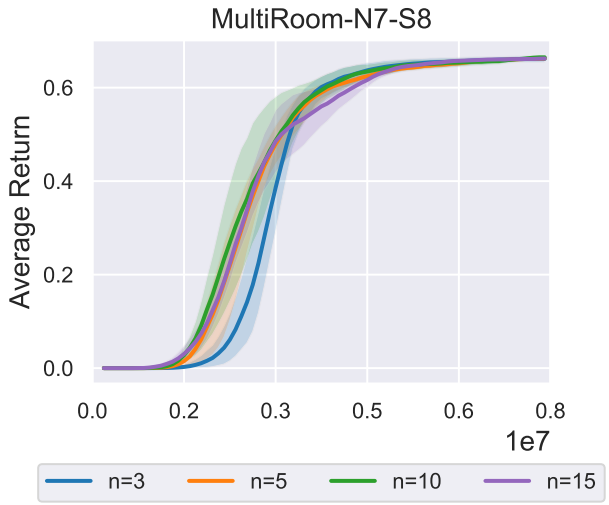


Figure C.2: Training performance of GoBI on MiniGrid MultiRoom-N7-S8 with different n randomly sampled actions per step.

for the states to be added to the episodic buffer, it will continue staying at the states to maximize its intrinsic reward. We tried to tune the similarity score threshold using grid search but still have not find a good hyper-parameter choice for it because the similarity score at different corners does not share a consistent value. Unlike EC, we can see from the figure that GoBI chooses not to visit the border of the room early on in training, as the information on the border are easily predictable from the information in the middle of the room.

C.4 Wallclock Training Time

Table C.5 shows the wall-clock time needed to train NovelD and our method for 10M MiniGrid environment steps. GoBI requires about 2x the wall-clock time needed to train NovelD for the same number of environment steps.

Algorithm	Wallclock Time (hours)
NovelD	5.46($\pm$0.058)
GoBI(Ours)	10.65($\pm$0.082)

Table C.5: Wallclock training time comparison in hours between NovelD and GoBI.

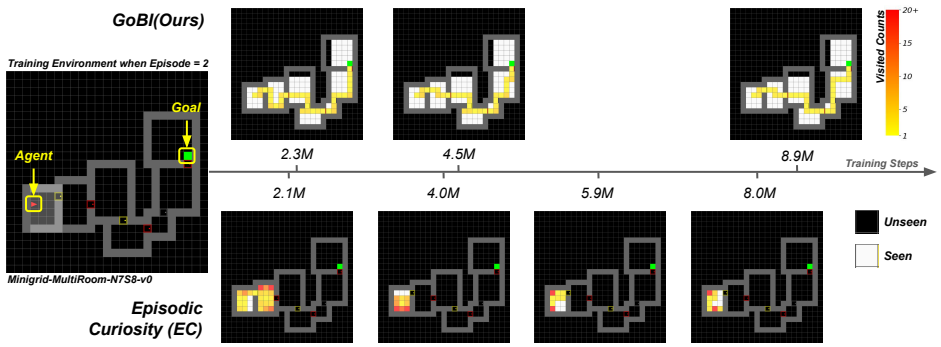


Figure C.3: Heatmaps of example trajectories at different training steps in MiniGrid-MultiRoom-N7-S8 environment.

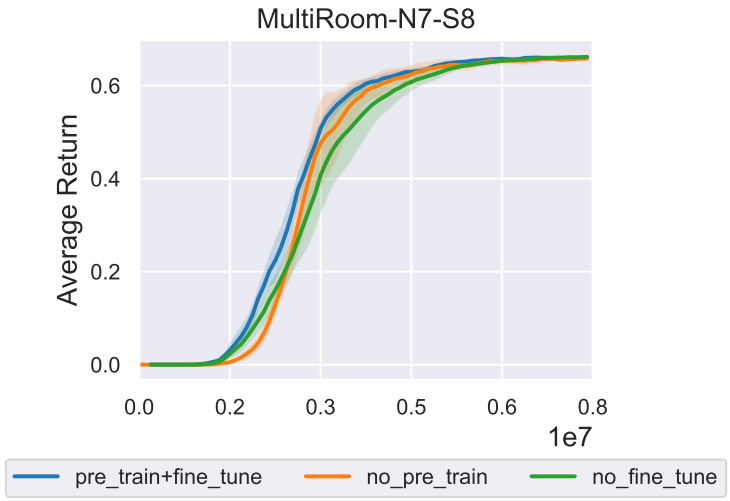


Figure C.4: Training performance comparison on MultiRoom-N7-S8 environment among 1) use a fixed pre-trained dynamics model, 2) use a pre-trained dynamics model, and fine-tune it online, 3) no pre-training, directly train the dynamics model together with policy training.

C.5 Dynamics Training

In Section 4.3, we report the results of applying a pre-trained forward dynamics for GoBI on MiniGrid. However, the forward dynamics model can also be trained together with policy training. In Figure C.4, we report the results of an ablation study on a MiniGrid environment of 3 settings: 1) use a pre-trained dynamics model, and keep it fixed when training the policy, 2) pre-train a dynamics model, and fine-tune it when training the policy, 3) no pre-training, directly train the dynamics model in an online manner. In summary, all 3 versions work similarly, but due to the fact that training the dynamics model online will add extra wall clock training time, we use option 1 in our

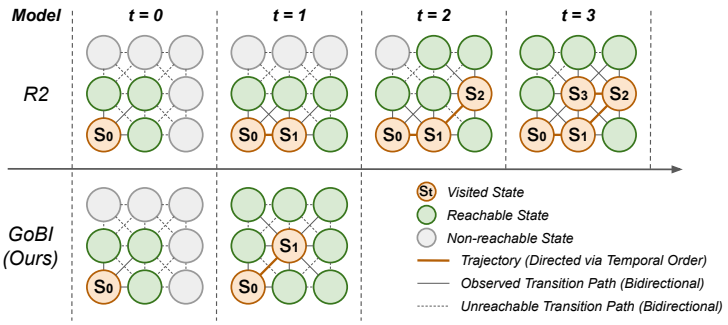


Figure C.5: An illustrative example of why R2 does not work well to encourage efficient exploration. In this example, our goal is to include all the states into the episodic buffer quickly. GoBI can move directly to the center in one step for maximum intrinsic reward, while R2 may choose to take extra steps for exploration since it mainly focuses on whether the episodic buffer expands or not.

main experiments.

C.6 Ablation Study Illustration

In this section, we provide an illustrative example of why only considering whether new states are added to the episodic buffer or not, i.e., R2 in Section 4.3.3, works significantly worse than our method. The example is shown in Figure C.5. If these 9 states are only a small part of the environment, we want a policy that explore this part as quickly as possible - mark all the states as reachable as quickly as possible. However, in order to maximize its step-wise intrinsic reward, an agent trained with R2 will go along the border to only add a few new states to the episodic buffer at a time, which wastes many unnecessary steps so is not beneficial for exploration.

APPENDIX D

Hierarchical Procedural Meta-Reasoning for Generalizable Multimodal Agents

D.1 Evaluation Settings

D.1.1 Evaluation Framework

Dynamic auto-evaluation. We evaluate all agents using a dynamic automatic evaluation framework based on realistic Android emulators set up in DigiData-Bench-Auto. Each episode runs inside an emulator with the target application installed and initiated with a pre-defined emulator image, and the environment state is reset at the beginning of each trial.

Success measurement with LLM-as-a-judge. As many tasks have open-ended natural language goals and multi-step UI interactions, by default, we determine task success using an LLM judge protocol with GPT-4o, which was established by the DigiData benchmark [143].

Action Space Agents interact with the environment through the following actions:

– `tap(x, y)`: Tap at screen location (x, y).
– `swipe(x1, y1, x2, y2)`: Swipe from (x_1, y_1) to (x_2, y_2).
– `type(text)`: Type a string into the currently focused input field.
– `navigate(option)`: System navigation actions with `option` $\in$ {`back`, `home`, `enter`}.
– `status(option)`: Terminate an episode with `option` $\in$ {`complete`, `impossible`}.

Algorithm 4 PAMAR Test-Time Execution with Meta-Reasoned Knowledge Acquisition

Require: Goal g, environment $\mathcal{E}$, procedure generator M_p, meta-reasoner M_r, grounded actor M_a, knowledge search tool $\mathsf{Search}(\cdot)$, knowledge processor $\mathsf{Accumulate}(\cdot)$

Observe initial screenshot $o_0 \leftarrow \mathcal{E}.\mathsf{reset}(g)$
Initialize procedure $p \leftarrow M_p(g, o_0)$
Initialize external knowledge buffer $K \leftarrow \emptyset$
for $t = 0, 1, 2, \ldots, T$ **do**
  Sufficiency check: $d_t \leftarrow M_r(g, o_{\leq t}, a_{<t}, p)$ $\{d_t \in \{\mathsf{continue}, \mathsf{search}\}\}$
  if $d_t = \mathsf{search}$ **then**
    Generate app-specific query $q_t \leftarrow \mathsf{QueryGen}(g, p, o_t)$
    Retrieve pages/screenshots $S_t \leftarrow \mathsf{Search}(q_t)$
    Extract knowledge $k_t \leftarrow \mathsf{Accumulate}(S_t)$
    Update buffer $K \leftarrow K \cup \{k_t\}$
  end if
  Procedure update: $p \leftarrow M_p(g, o_t, a_{<t}, p, K)$
  Grounded action: $a_t \leftarrow M_a(g, o_t, a_{<t}, p)$
  Execute a_t, observe next screenshot $o_{t+1} \leftarrow \mathcal{E}.\mathsf{step}(a_t)$
  if success or termination condition then
    break
  end if
end for

D.1.2 Datasets

We evaluate PAMAR along three complementary generalization axes.

D.1.2.1 Task and application generalization.

We use DigiData-Bench-Auto [143], which evaluates generalization across tasks and applications by grouping evaluation apps into *seen*, *familiar*, and *novel* categories with respect to the training set. We follow the standard protocol of [143] for environment initialization, task definitions, and success evaluation.

D.1.2.2 Application version update adaptation.

To isolate distribution shift caused by post-training application updates, we construct DigiData-AppUpdate. Starting from the DigiData training task pool, we identify 61 goals on *seen* applications whose original successful trajectories no longer execute reliably due to version changes. We then re-evaluate the same goals on the updated app versions and group update-induced failures into two common patterns: (i) functionality deprecation,

where a previously available feature is removed or renamed, making the original workflow infeasible; and (ii) workflow/UI drift, where the task remains supported but the interface or navigation changes (e.g., layout changes, deeper menus, or relocated controls). Figure 5.2 shows representative examples of both patterns.

D.1.2.3 Single-to-multi app generalization (DigiData-MultiApp).

To evaluate whether agents trained only on single-app trajectories can generalize to multi-app workflows, we construct DigiData-MultiApp, a benchmark of 20 tasks that require composing actions across two applications. Each task requires (i) obtaining a piece of information or completing a subgoal in a source app, and (ii) switching to a target app to perform a follow-up action conditioned on the result from the source app. Agents are trained on single-app objectives only and never observe multi-app demonstrations during training.

D.2 Procedure-Aware Multimodal Agent with Meta-Reasoning

D.2.1 Model Training

We fine-tune PLM-8B using supervised fine-tuning (SFT). Training is conducted in two stages. In the first stage, we train the model for 60K iterations on base data without procedures. In the second stage, we continue training for an additional 15K iterations on a mixture of data with and without procedures. Unless otherwise specified, we use a learning rate of 2×10^{-5} and a batch size of 2 samples per GPU. We fine-tune the full model, including the vision encoder and adaptation layers. We use an image tile size of 448×448 and limit each screenshot to at most four tiles. Training is performed on 128 NVIDIA A100 GPUs.

D.2.2 Procedure Synthesis

This section details the procedure induction and language augmentation pipeline used to construct procedural supervision from expert demonstrations. In DigiData [143], each demonstration is a length-T trajectory of observation–action pairs $\{(o_t, a_t)\}_{t=1}^{T}$, where o_t is a screenshot and a_t is an atomic UI action (e.g., tap, swipe, type). Our goal is to

convert such low-level trajectories into compact natural-language procedures that can be used to condition the grounded executor during training.

D.2.2.1 Transition-Grounded Step Extraction

We first convert each atomic action into a short natural-language step grounded in the observed UI transition. For each timestep t, we prompt an LLM with: (i) the pre-action screenshot o_{t-1}, (ii) the executed action a_t (including tap/swipe coordinates), and (iii) the post-action screenshot o_t. The model outputs a structured summary of the transition; we extract from this output a concise action-intent description and denote it as a per-timestep description d_t. We use Prompt STEP SUMMARIZATION PROMPT for this purpose, which is based on the Chain-of-Thought generation prompt from DigiData [143].

Prompt D.2.1: STEP SUMMARIZATION PROMPT

You are an expert in summarizing the progress of an Android navigation agent, whose role is to help a human user navigate the Android phone to complete a goal. The agent takes step-by-step phone actions like clicking, swiping, typing, navigating home, navigating back, and ending.

The agent's action at this step is {`action`}. If the action involves tapping or long press, the tap location is shown on screen A with a hollow green circle. If the action involves swiping, the start and end points of the swipe are shown on Screen A with a green arrow. {`xml_string`}

First summarize screen A, screen B. Focus on:

- What key elements (e.g., icons, texts, buttons, navigation bars, images, forms and input field, pop-ups, switches and toggles etc) are there in each screenshot and where are they (e.g., top-left, bottom-right etc).
- Do NOT mention the terms "screen A" or "screen B" in your response. Instead, refer to them as the screenshot.

Next describe the agent's action. Make it short, in one sentence, and use future tense. Focus on:

- What does the agent do to get from A to B. If the action involves tapping, infer which key element is tapped based on the action, hollow green circle location and the context of the screenshot. If the action involves swiping,

infer the swipe direction based on the action, green arrow and the context of the screenshot.

- Do NOT mention the terms "hollow green circle" or "green arrow" in your response.

Then summarize the changes from A to B. Use future tense. Focus on:

- What are the changes from A to B related to the goal. Pay attention to subtle changes including altered or new key elements.
- Do NOT mention the terms "screen A" or "screen B" in your response. Refer to screen A as the current screenshot and screen B as the next screenshot.

Finally provide a reasoning on why does the agent take this action. Use future tense. Focus on:

- What are the reasons for the agent to take this action? Why does this action help the agent to achieve the goal?

Be concise and comprehensive. Only mention key elements which are related to the goal and ignore the irrelevant ones.

Use the first-person pronoun "I" as if you are the agent.

Your answer should be in the following format, ensuring each section is no more than 100 words:

screen A: <summary of screen A>

screen B: <summary of screen B>

action description: <action description>

changes: <changes from A to B>

reason: <why took this action>

D.2.2.2 Step Merging and Normalization

The per-timestep descriptions $\{d_t\}$ are fine-grained and often contain redundant retries or micro-actions (e.g., repeated scrolls or taps). We therefore merge consecutive descriptions

into a shorter sequence of higher-level steps $\{s_k\}_{k=1}^{K}$ using Android-specific heuristics:

- **Action-form normalization.** Before comparing or merging step descriptions, we canonicalize their surface form by removing tense/inflectional variants, common prepositions, and personal pronouns that do not affect intent. This yields a normalized representation that better reflects the underlying UI intent.
- **Tap equivalence under stable observations.** For tap actions, we treat two consecutive steps as equivalent if (i) the pre-action observations are identical and (ii) the tap coordinates are within a small pixel radius. Intuitively, repeated taps on the same screen at approximately the same location typically correspond to the same intended UI interaction.
- **Action consolidation.** Using the normalized action forms and the tap-equivalence rule above, we merge consecutive steps with the same inferred intent into a single abstract instruction. For repeated actions, we express repetition explicitly (e.g., *Tap X a few times* or *Scroll down several times*).

D.2.2.3 Condition-Guided Abstraction

To better reflect how humans express procedures, we further rewrite action-centric steps into intent- and condition-guided subgoals whenever applicable. For example, we replace fixed-count scroll instructions (e.g., *Swipe up twice*) with state-dependent forms (e.g., *Scroll up until you see the desired element*). This transformation encodes both the intent and an explicit stopping condition, improving robustness to UI variability and reducing sensitivity to exact action counts. We also standardize completion markers (e.g., issue *status(complete) to complete the task*) to produce more consistent procedure structure.

D.2.2.4 Procedure Language Augmentation

To improve robustness to linguistic variation at test time, we generate paraphrases for each induced procedure. Concretely, we prompt an LLM to rewrite the merged procedure into alternative natural-language forms while preserving its semantics. In our implementation, we provide the model with the sequence of step descriptions and ask it to produce a more narrative, high-level version of the same procedure.

Prompt D.2.2: NARRATIVE PROCEDURE REWRITING PROMPT

You are an expert in summarizing the progress of an Android navigation agent, whose role is to help a human user navigate the Android phone to complete a goal. The agent takes step-by-step phone actions like clicking, swiping, typing, navigating home, navigating back, and ending.

You are given a trajectory that aims to achieve the goal of "{goal}". There are {total_steps} steps in this trajectory and you are currently at step {step_index}. For each step, you are provided with a description of the screenshot before agent's action and the agent's action.

First, understand the progress by summarizing what has already happened based on the step-by-step descriptions below of what the agent has done before step {step_index}:

{history_string}

Next, summarize what will happen as a procedure based on the step-by-step descriptions below of what the agent will do in and after step {step_index}:

{future_string}

Please follow these guidelines when providing your response:

- Do NOT mention specific step numbers.
- Follow the step-by-step descriptions closely when summarizing the progress and the procedure.
- Use the first-person pronoun "I" as if you are the agent.

Your answer should be in the following format, with each section limited to 100 words or fewer:

Summary: <summary of all previous steps>

procedure: <procedure of all future steps (including current one)>

D.2.3 Procedure Generation

The procedure generator in this chapter is based on GPT-4o unless otherwise specified. Prompts for procedure generation and procedure update are shown below.

Prompt D.2.3: PROCEDURE GENERATOR

You are an expert in creating step-by-step procedures for Android navigation tasks. Your role is to provide complete, detailed instructions that would help someone who has never done this task before to accomplish their goal from start to finish. You must first reason through the task by analyzing what needs to be done, then provide a comprehensive step-by-step procedure.

Please follow these guidelines when providing your response:

- CRITICAL: Each step must correspond to exactly ONE atomic action. The agent can only perform one of these actions per step: <action space description >
- Never let the agent log in or update apps.
- UI ELEMENT DESCRIPTION: The bottom-left corner is reserved for the 'Navigate Back' button. When referring to other elements near the bottom-left, describe their appearance only (icon/text), or explicitly say 'near the corner but NOT the navigate back button'.
- RESOLVE CONDITIONS:

Please provide a step-by-step procedure to accomplish the given goal from start to finish, as if creating instructions for someone who has never done this task before. The procedure should start from the beginning of the workflow when you open the app.

Example 1:

Goal: Use Settings app, Disconnect from currently connected Wi-Fi networks

Step-by-Step Procedure:

1. Swipe up a few times until you see Settings icon.
2. Tap on the Settings icon.
3. Tap on "Internet".
4. Tap on the Wi-Fi toggle switch to turn it off.
5. Tap on "Done".
6. Issue "status(complete)" to stop the task.

Example 2:

...

Example 3:

...

Now, provide the step-by-step procedure for the following goal:

Goal: "goal"

Your answer should be in the following format:

Step-by-Step Procedure:
1. [First step]
2. [Second step]
3. [Third step]
...
N. [Final step]

Prompt D.2.4: PROCEDURE UPDATE PROMPT

Review the current screenshot and action history to refine the existing step-by-step procedure if needed.
IMPORTANT: Even when updating the plan based on the current screen, you must provide a COMPLETE procedure as general knowledge - starting from the very beginning when the user first opens the app. Do NOT provide a partial plan that only covers remaining steps. The procedure should serve as a comprehensive guide that works for anyone attempting this task from scratch.
Instructions:
Compare the current screen with the previous plan. Make minor adjustments only if:
- A UI element name differs from expected (e.g., `"My Account"` instead of `"Account"`)
- The screen state reveals a different path is needed
If the plan is still accurate, return it unchanged.
First, reason through what adjustments (if any) are needed based on the current screen state.
- Although you should generate the whole procedure, pay attention to the current observation/history to correct any existing errors.
- Ignore any update/log in steps.
- UI ELEMENT DESCRIPTION: The bottom-left corner is reserved for the 'Navigate Back' button. When referring to other elements near the bottom-left, describe their appearance only (icon/text), or explicitly say 'near the corner but NOT the navigate back button'.
- The last step must be `Issue "status(...)" to stop the task`
Here are some examples of step-by-step procedures we want from you, notice that it should always be starting from the very beginning:
Example 1:

Goal: Use Settings app, Disconnect from currently connected Wi-Fi networks
Step-by-Step Procedure:
1. Swipe up a few times until you see Settings icon.
2. Tap on the Settings icon.
3. Tap on `"Internet"`.
4. Tap on the Wi-Fi toggle switch to turn it off.
5. Tap on `"Done"`.
6. Issue `"status(complete)"` to stop the task.
Example 2: ...
Example 3: ... Now, for the following goal, first provide your reasoning about any needed adjustments, then provide the complete step-by-step procedure (starting from app launch).
Goal: `"{goal}"`
Previous Plan:
{`prev_plan`}
Action History:
{`action_history`}
Your answer MUST follow this exact format, DO NOT put anything after the last step of the procedure:
Reasoning:
[Your analysis of the current screen and any adjustments needed]
Step-by-Step Procedure:
1. [First step - should typically involve opening/launching the app]
2. [Second step]
3. [Third step]
...
N. [Final step]

D.2.4 Meta Reasoner

The meta reasoning prompt is shown below.

Prompt D.2.5: META REASONER

You are an expert Android task execution assistant. Your role is to analyze the current task progress by carefully tracking where the agent is in its plan, and decide whether external knowledge is needed to proceed effectively.
You will be given:
1. The goal/task to accomplish
2. The plan you generated last step (step-by-step procedure)
3. The action history (what has been done so far)
4. The most recent two screenshots
Your job is to determine if the agent is on the right track or if external knowledge is needed.
Analyze the current task execution and decide if external knowledge is needed.
Goal: {`goal`}
Current Plan:
{`current_plan`}
Action History:
{`action_history`}
Instructions - Follow this reasoning process step by step:
Step 1: Locate Current Position in Plan
- Compare the current observation and action history against the current plan and identify which step number the agent has just completed or is currently executing.
Step 2: Verify Previous Action Outcome
- Compare the current screen with the previous screen (you have access to the most recent images).
- Did the last action executed match the plan?
- Did that action produce the expected change in the UI?
Step 3: Assess Next Step & Plan Validity
- Is the UI element or action required for the *next* step visible/available on the current screen?
- **If YES:** The plan is working. ⇒ **CONTINUE**
- **If NO:** Analyze WHY:
- **Case A (Minor Mismatch / Easy Fix):** The specific element is missing, BUT there is an obvious alternative that achieves the same sub-goal (e.g., button is named slightly differently, or there's a clear path forward). The agent's general

knowledge is correct, just needs a minor plan adjustment. ⇒ **CONTINUE**
- **Case B (Wrong Plan / Knowledge Gap):** The previous action didn't work as expected (e.g., tapped a button but nothing happened, or went to a wrong screen), OR the next step is missing and there is NO obvious path forward. This implies the agent's understanding of the procedure is incorrect. ⇒ **SEARCH**
Your response MUST follow this exact format:
Reasoning:
Step 1 - Position: [Where is the agent in the plan?]
Step 2 - Prev Action Check: [Did the last action work as expected? Explain based on screen changes.]
Step 3 - Analysis: [Is the next step achievable? If not, is it a minor mismatch (CONTINUE) or a wrong plan (SEARCH)?]
Decision: [CONTINUE or SEARCH]

APPENDIX E

Against Epistemic Shortcuts: Evidence-Grounded Verification for LLM Agents

E.1 The PackAudit Environment

E.1.1 Task Definition

We design a text-based toy environment called PackAudit to study epistemic shortcuts in LLM agents. In each episode, the agent is tasked with auditing a simulated software package represented as a small repository tree.

The agent is given three compliance requirements, such as having no copyleft license, having a README quality score of at least 8, and having zero known vulnerabilities. It must then decide whether to approve or reject the package. Each requirement can be verified using exactly one evidence file, and the three evidence files are distributed across different directories in the repository. Solving the task therefore requires the agent to navigate the repository and gather sufficient evidence to make a final decision.

E.1.2 Updated and Outdated Summary Settings

To induce potential epistemic shortcut behavior, the root dashboard also displays a *Quick Scan Summary (Cached)*. The summary contains an overall package status, one cached field for each requirement, and a staleness timestamp. Because this summary may become outdated, we test whether agents are misled into exhibiting epistemic shortcut behavior under two different settings.

In the **outdated summary** setting, the overall verdict in the cached summary is intentionally flipped relative to the ground truth, and the summary has a staleness timestamp

of 30 days. An agent must therefore inspect the underlying evidence files rather than rely on the cached summary.

In the **updated summary** setting, the cached summary displayed on the root dashboard is consistent with the underlying evidence and has a staleness timestamp of 1 second. Relying on the summary is therefore sufficient to reach the correct final decision. This setting serves as a baseline for evaluating the agent's general task-solving ability.

E.1.3 Observation, Action Space, and Reward

Observation Space. Observations consist of three page types: the repository root dashboard, folder listings, and file views. The cached summary is displayed only on the root dashboard.

Action Space. The agent can take the following actions:

- `open[folder]`: open a directory;
- `view[file]`: inspect the contents of a file;
- `back`: return to the previous directory;
- `approve/reject`: approve or reject the package and terminate the episode.

Invalid actions leave the navigation state unchanged and return an error message together with the current observation.

Termination and Reward. An episode terminates when the agent submits an `approve` or `reject` message. The agent receives a reward of 1 if its decision matches the ground-truth conjunction of the three requirements, and 0 otherwise. The episode horizon is set to 15 steps.

BIBLIOGRAPHY

[1] Google Flights. https://www.google.com/travel/flights. Accessed: 2024-05-21.

[2] Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. arXiv preprint arXiv:1703.01732, 2017.

[3] Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Wang. Agent s: An open agentic framework that uses computers like a human. In International Conference on Learning Representations, volume 2025, pages 22924–22946, 2025.

[4] Per Andersen, Richard Morris, David Amaral, Tim Bliss, and John O'Keefe. The hippocampus book. Oxford university press, 2006.

[5] Chinmaya Andukuri, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah Goodman. STar-GATE: Teaching language models to ask clarifying questions. In First Conference on Language Modeling, 2024.

[6] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martin Arjovsky, Alexander Pritzel, Andrew Bolt, and Charles Blundell. Never give up: Learning directed exploration strategies. In International Conference on Learning Representations, 2020.

[7] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. arXiv preprint arXiv:1612.03801, 2016.

[8] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. arXiv preprint arXiv:1606.04460, 2016.

[9] SRK Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a monte-carlo framework. Journal of Artificial Intelligence Research, 43:661–704, 2012.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[11] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In Conference on Robot Learning, pages 287–318. PMLR, 2023.

[12] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In International Conference on Learning Representations, 2019.

[13] Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Guozhi Wang, Dingyu Zhang, Shuai Ren, and Hongsheng Li. Amex: Android multi-annotation expo dataset for mobile gui agents. In Findings of the Association for Computational Linguistics: ACL 2025, page 2138–2156. Association for Computational Linguistics, 2025.

[14] Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Goal-conditioned reinforcement learning with imagined subgoals. In International conference on machine learning, pages 1430–1440. PMLR, 2021.

[15] Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. Advances in Neural Information Processing Systems, 37:589–631, 2024.

[16] Zhixun Chen, Ming Li, Yuxuan Huang, Yali Du, Meng Fang, and Tianyi Zhou. Atlas: Agent tuning via learning critical steps. In Findings of the Association for Computational Linguistics: ACL 2025, pages 25334–25349, 2025.

[17] Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. CoRR, abs/2306.13831, 2023.

[18] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

[19] Jang Hyun Cho, Andrea Madotto, Effrosyni Mavroudi, Triantafyllos Afouras, Tushar Nagarajan, Muhammad Maaz, Yale Song, Tengyu Ma, Shuming Hu, Suyog Jain, et al. Perceptionlm: Open-access data and models for detailed visual understanding. Advances in Neural Information Processing Systems, 38:23475–23537, 2026.

[20] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In International conference on machine learning, pages 2048–2056. PMLR, 2020.

[21] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In International Conference on Machine Learning, pages 1282–1289. PMLR, 2019.

[22] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. Advances in Neural Information Processing Systems, 36:28091–28114, 2023.

[23] 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.

[24] Zane Durante, Ran Gong, Bidipta Sarkar, Naoki Wake, Rohan Taori, Paul Tang, Shrinidhi Lakshmikanth, Kevin Schulman, Arnold Milstein, Hoi Vo, et al. An interactive agent foundation model. In Proceedings of the Computer Vision and Pattern Recognition Conference, pages 3652–3662, 2025.

[25] Howard Eichenbaum. The role of the hippocampus in navigation is memory. Journal of neurophysiology, 117(4):1785–1796, 2017.

[26] Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In International Conference on Machine Learning, pages 15419–15462. PMLR, 2025.

[27] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In International conference on machine learning, pages 1407–1416. PMLR, 2018.

[28] Yannis Flet-Berliac, Johan Ferret, Olivier Pietquin, Philippe Preux, and Matthieu Geist. Adversarially guided actor-critic. In International Conference on Learning Representations, 2021.

[29] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In International conference on machine learning, pages 1515–1528. PMLR, 2018.

[30] Daniel Freeman, David Ha, and Luke Metz. Learning to predict without looking ahead: World models without forward prediction. Advances in Neural Information Processing Systems, 32, 2019.

[31] Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma GongQue, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. Agentrefine: Enhancing agent generalization through refinement tuning. In The Thirteenth

International Conference on Learning Representations, 2025.

[32] Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. Advances in Neural Information Processing Systems, 37:119919–119948, 2024.

[33] Yao Fu, Run Peng, and Honglak Lee. Go beyond imagination: maximizing episodic reachability with world models. In International Conference on Machine Learning, pages 10405–10420. PMLR, 2023.

[34] Yao Fu, Shengyi Qian, Pierluca D'Oro, Fanyi Xiao, Honglak Lee, Joseph Tighe, and Manchen Wang. Hierarchical procedural meta-reasoning for generalizable multimodal agents. In Forty-third International Conference on Machine Learning, 2026.

[35] Yao Fu, Ran Qiu, Xinhe Wang, Jacob Sansom, Sathvika Ayyappa Prabhu, Huijie Tang, Jaekyeom Kim, Sungryull Sohn, and Honglak Lee. Beyond blind following: Evaluating robustness of llm agents under imperfect guidance. In Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6591–6618, 2026.

[36] Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models. In International Conference on Learning Representations, volume 2024, pages 29528–29558, 2024.

[37] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In International Conference on Machine Learning, pages 10764–10799. PMLR, 2023.

[38] Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In The Twelfth International Conference on Learning Representations, 2024.

[39] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models, 2024. Preprint.

[40] Li Gu, Zihuan Jiang, Zhixiang Chi, Huan Liu, Ziqiang Wang, Yuanhao Yu, Glen Berseth, and Yang Wang. Generalization in online reinforcement learning for mobile agents, 2026.

[41] Priyanshu Gupta, Shashank Kirtania, Ananya Singha, Sumit Gulwani, Arjun Radhakrishna, Gustavo Soares, and Sherry Shi. Metareflection: Learning instructions for language agents using past reflections. In Proceedings of the 2024 Conference

on Empirical Methods in Natural Language Processing, pages 8369–8385, 2024.

[42] Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. Understanding HTML with large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, Findings of the Association for Computational Linguistics: EMNLP 2023, pages 2803–2821, Singapore, December 2023. Association for Computational Linguistics.

[43] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. Advances in neural information processing systems, 31, 2018.

[44] David Ha and Jürgen Schmidhuber. World models. arXiv preprint arXiv:1803.10122, 2018.

[45] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning, pages 1861–1870. Pmlr, 2018.

[46] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In International Conference on Learning Representations, 2020.

[47] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. arXiv preprint arXiv:2301.04104, 2023.

[48] Austin W Hanjie, Victor Y Zhong, and Karthik Narasimhan. Grounding language to entities and dynamics for generalization in reinforcement learning. In International Conference on Machine Learning, pages 4051–4062. PMLR, 2021.

[49] Elad Hazan, Sham Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. In International Conference on Machine Learning, pages 2681–2691. PMLR, 2019.

[50] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6864–6890, 2024.

[51] Qianyu He, Jie Zeng, Qianxi He, Jiaqing Liang, and Yanghua Xiao. From complex to simple: Enhancing multi-constraint complex instruction following ability of large language models. In Findings of the Association for Computational Linguistics: EMNLP 2024, pages 10864–10882, 2024.

[52] Juyeon Heo, Christina Heinze-Deml, Oussama Elachqar, Kwan Ho Ryan Chan, Shirley Ren, Andrew Miller, Udhyakumar Nallasamy, and Jaya Narain. Do llms“know”internally when they follow instructions? In International Conference on Learning Representations, volume 2025, pages 81339–81357, 2025.

[53] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In International Conference on Learning Representations, 2022.

[54] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. ACM Transactions on Information Systems, 43(2):1–55, January 2025.

[55] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In International Conference on Machine Learning, pages 9118–9147. PMLR, 2022.

[56] Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. Discoveryworld: A virtual environment for developing and evaluating automated scientific discovery agents. Advances in Neural Information Processing Systems, 37:10088–10116, 2024.

[57] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In International Conference on Learning Representations, volume 2024, pages 54107–54157, 2024.

[58] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In Second Conference on Language Modeling, 2025.

[59] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. Challenges and applications of large language models, 2023.

[60] 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, 2024.

[61] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. Position:

Llms can't plan, but can help planning in llm-modulo frameworks. In Forty-first International Conference on Machine Learning, 2024.

[62] Kirandeep Kaur, Xingda Lyu, and Chirag Shah. Position: Knowing isn't understanding: Re-grounding generative proactivity with epistemic and behavioral insight. In Forty-third International Conference on Machine Learning Position Paper Track, 2026.

[63] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In 2016 IEEE conference on computational intelligence and games (CIG), pages 1–8. IEEE, 2016.

[64] Saksham Khatwani, He Cheng, Majid Afshar, Dmitriy Dligach, and Yanjun Gao. Brittleness and promise: Knowledge graph based reward modeling for diagnostic reasoning. ArXiv, abs/2509.18316, 2025.

[65] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. Advances in Neural Information Processing Systems, 36:39648–39677, 2023.

[66] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 881–905, 2024.

[67] Jing Yu Koh, Stephen Marcus McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. Transactions on Machine Learning Research, 2025.

[68] J Zico Kolter and Andrew Y Ng. Near-bayesian exploration in polynomial time. In Proceedings of the 26th annual international conference on machine learning, pages 513–520, 2009.

[69] Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In Neural Information Processing Systems, 2016.

[70] Ananya Kumar, Jiahui Yu, John Hallman, Michelle Pokrass, et al. Introducing gpt-4.1 in the api, April 2025.

[71] Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Aaron Jiaxun Li, Soheil Feizi, and Himabindu Lakkaraju. Certifying LLM safety against adversarial prompting. In First Conference on Language Modeling, 2024.

[72] Daeyong Kwon, Soyoung Yoon, and Seung won Hwang. Safe: An llm-as-verifier framework for evidence-grounded multi-hop reasoning, 2026.

[73] Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms. arXiv preprint arXiv:2406.18629, 2024.

[74] Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. Advances in neural information processing systems, 33:19884–19895, 2020.

[75] Hugo Laurençon, Léo Tronchon, Matthieu Cord, and Victor Sanh. What matters when building vision-language models? Advances in Neural Information Processing Systems, 37:87874–87907, 2024.

[76] Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. arXiv preprint arXiv:1906.05274, 2019.

[77] Mukai Li, Qingcheng Zeng, Tianqing Fang, Zhenwen Liang, Linfeng Song, Qi Liu, Haitao Mi, and Dong Yu. Verified critical step optimization for LLM agents. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens, editors, Findings of the Association for Computational Linguistics: ACL 2026, pages 39627–39639, San Diego, California, United States, July 2026. Association for Computational Linguistics.

[78] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile ui action sequences. In Proceedings of the 58th annual meeting of the association for computational linguistics, pages 8198–8210, 2020.

[79] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In 2023 IEEE International conference on robotics and automation (ICRA), pages 9493–9500. IEEE, 2023.

[80] Bingqian Lin, Yi Zhu, Yanxin Long, Xiaodan Liang, Qixiang Ye, and Liang Lin. Adversarial reinforced instruction attacker for robust vision-language navigation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(10):7175–7189, 2021.

[81] Zongyu Lin, Yao Tang, Xingcheng Yao, Da Yin, Ziniu Hu, Yizhou Sun, and Kai-Wei Chang. QLASS: Boosting language agent inference via q-guided stepwise search. In Forty-second International Conference on Machine Learning, 2025.

[82] Anthony Zhe Liu, Jongwook Choi, Sungryull Sohn, Yao Fu, Jaekyeom Kim, Dong-

Ki Kim, Xinhe Wang, Jaewon Yoo, and Honglak Lee. Skillact: Using skill abstractions improves llm agents. In ICML 2024 Workshop on LLMs and Cognition, 2024.

[83] Anthony Zhe Liu, Xinhe Wang, Jacob Sansom, Yao Fu, Jongwook Choi, Sungryull Sohn, Jaekyeom Kim, and Honglak Lee. Interactive and expressive code-augmented planning with large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 20330–20354, Vienna, Austria, July 2025. Association for Computational Linguistics.

[84] Hao Liu and Pieter Abbeel. Behavior from the void: Unsupervised active pre-training. Advances in Neural Information Processing Systems, 34:18459–18473, 2021.

[85] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In The Twelfth International Conference on Learning Representations, 2024.

[86] Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Song XiXuan, Yifan Xu, Shudan Zhang, Hanyu Lai, Jiadai Sun, Xinyue Yang, et al. Visualagentbench: Towards large multimodal models as visual foundation agents. In International Conference on Learning Representations, volume 2025, pages 95650–95707, 2025.

[87] Xuannan Liu, Xiao Yang, Zekun Li, Peipei Li, and Ran He. Agenthallu: Benchmarking automated hallucination attribution of llm-based agents, 2026.

[88] Lajanugen Logeswaran, Yao Fu, Moontae Lee, and Honglak Lee. Few-shot subgoal planning with language models. In NAACL: HLT, 2022.

[89] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In ACL, 2022.

[90] Yougang Lyu, Xiaoyu Zhang, Lingyong Yan, Maarten de Rijke, Zhaochun Ren, and Xiuying Chen. Deepshop: A benchmark for deep research shopping agents. arXiv preprint arXiv:2506.02839, 2025.

[91] Ziqiao Ma, Benjamin VanDerPloeg, Cristian-Paul Bara, Yidong Huang, Eui-In Kim, Felix Gervits, Matthew Marge, and Joyce Chai. DOROTHIE: Spoken dialogue for handling unexpected situations in interactive autonomous driving agents. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, Findings of the

Association for Computational Linguistics: EMNLP 2022, pages 4800–4822, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[92] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. Advances in neural information processing systems, 36:46534–46594, 2023.

[93] Jianbiao Mei, Tao Hu, Daocheng Fu, Licheng Wen, Xuemeng Yang, Rong Wu, Pinlong Cai, Xinyu Cai, Xing Gao, Yu Yang, Chengjun Xie, Botian Shi, Yong Liu, and Yu Qiao. O$ˆ2$-searcher: A searching-based agent model for open-domain open-ended question answering. Transactions on Machine Learning Research, 2026.

[94] Jacob Menick, Kevin Lu, Shengjia Zhao, Eric Wallace, Hongyu Ren, Haitang Hu, Nick Stathas, and Felipe Petroski Such. Gpt-4o mini: advancing cost-efficient intelligence, July 2024.

[95] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In EMNLP, 2022.

[96] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937. PMLR, 2016.

[97] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[98] John Murzaku, Zifan Liu, Md Mehrab Tanjim, Vaishnavi Muppala, Xiang Chen, and Yunyao Li. Eclair: Enhanced clarification for interactive responses. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, pages 28864–28870, 2025.

[99] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In Neural Information Processing Systems, 2018.

[100] Masashi Okada, Norio Kosaka, and Tadahiro Taniguchi. Planet of the bayesians: Reconsidering and improving deep planning network by incorporating bayesian inference. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5611–5618. IEEE, 2020.

[101] OpenAI. Hello GPT-4. https://openai.com/index/hello-gpt-4o/, 2024. Accessed:

2024-05-16.

[102] Jie Ouyang, Tingyue Pan, Mingyue Cheng, Ruiran Yan, Yucong Luo, Jiaying Lin, and Qi Liu. Hoh: A dynamic benchmark for evaluating the impact of outdated information on retrieval-augmented generation. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6036–6063, 2025.

[103] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. Advances in neural information processing systems, 35:27730–27744, 2022.

[104] Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. BALROG: Benchmarking agentic LLM and VLM reasoning on games. In The Thirteenth International Conference on Learning Representations, 2025.

[105] Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. Webcanvas: Benchmarking web agents in online environments, 2024.

[106] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. arXiv preprint arXiv:2205.12255, 2022.

[107] Simone Parisi, Victoria Dean, Deepak Pathak, and Abhinav Gupta. Interesting object, curious agent: Learning task-agnostic exploration. Advances in Neural Information Processing Systems, 34, 2021.

[108] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In International conference on machine learning, pages 2778–2787. PMLR, 2017.

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

[110] Anselm Paulus, Arman Zharmagambetov, Chuan Guo, Brandon Amos, and Yuandong Tian. Advprompter: Fast adaptive adversarial prompting for LLMs. In Forty-second International Conference on Machine Learning, 2025.

[111] Chris Paxton, Yonatan Bisk, Jesse Thomason, Arunkumar Byravan, and Dieter Foxl. Prospection: Interpretable plans from language by predicting the future. In 2019 International Conference on Robotics and Automation (ICRA), pages 6942–6948. IEEE, 2019.

[112] PixyMoon. CUTERPG pack. https://pixymoon.itch.io/cuterpg, n.d. Accessed: 2025-06-23.

[113] Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models. In Findings of the Association for Computational Linguistics: NAACL 2024, pages 4226–4252, 2024.

[114] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In International Conference on Machine Learning, pages 2827–2836. PMLR, 2017.

[115] Pygame. Pygame. https://www.pygame.org/, 2024. Accessed: 2024-06-05.

[116] Cheng Qian, Bingxiang He, Zhong Zhuang, Jia Deng, Yujia Qin, Xin Cong, Zhong Zhang, Jie Zhou, Yankai Lin, Zhiyuan Liu, et al. Tell me more! towards implicit user intention understanding of language model driven agents. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1088–1113, 2024.

[117] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In International Conference on Learning Representations, volume 2024, pages 9695–9717, 2024.

[118] Qwen Team. Qwen3: Think deeper, act faster, April 2025.

[119] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. Advances in neural information processing systems, 36:53728–53741, 2023.

[120] Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. In International Conference on Learning Representations, 2020.

[121] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Androidinthewild: A large-scale dataset for android device control. Advances in Neural Information Processing Systems, 36:59708–59728, 2023.

[122] Matthew Renze and Erhan Guven. The benefits of a concise chain of thought on problem-solving in large language models. In 2024 2nd International Conference on Foundation and Large Language Models (FLLM), page 476–483. IEEE, November 2024.

[123] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In International conference on machine learning, pages 4344–4353. PMLR, 2018.

[124] Junha Roh, Chris Paxton, Andrzej Pronobis, Ali Farhadi, and Dieter Fox. Conditional driving from natural language instructions. In Conference on Robot Learning, pages 540–551. PMLR, 2020.

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

[126] Nikolay Savinov, Anton Raichuk, Damien Vincent, Raphael Marinier, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. In International Conference on Learning Representations, 2019.

[127] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. Advances in neural information processing systems, 36:68539–68551, 2023.

[128] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[129] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In International Conference on Machine Learning, pages 8583–8592. PMLR, 2020.

[130] Younggyo Seo, Lili Chen, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. State entropy maximization with random encoders for efficient exploration. In Marina Meila and Tong Zhang, editors, Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pages 9443–9454. PMLR, 18–24 Jul 2021.

[131] Jiaqi Shao, Yuxiang Lin, Munish Prasad Lohani, Yufeng Miao, and Bing Luo. Do llm agents know how to ground, recover, and assess? a benchmark for epistemic competence in information-seeking agents, 2025.

[132] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In Thirty-seventh Conference on Neural Information Processing Systems, 2023.

[133] Segev Shlomov, Ben wiesel, Aviad Sela, Ido Levy, Liane Galanti, and Roy Abitbol. From grounding to planning: Benchmarking bottlenecks in web agents, 2024.

[134] 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 The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[135] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In ICLR, 2021.

[136] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 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, 2023.

[137] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 7584–7600, 2024.

[138] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. arXiv preprint arXiv:1507.00814, 2015.

[139] Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. Journal of Computer and System Sciences, 74(8):1309–1331, 2008.

[140] Zhe Su, Xuhui Zhou, Sanketh Rangreji, Anubha Kabra, Julia Mendelsohn, Faeze Brahman, and Maarten Sap. Ai-liedar: Examine the trade-off between utility and truthfulness in llm agents. In Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 11867–11894, 2025.

[141] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. Advances in neural information processing systems, 36:58202–58245, 2023.

[142] Yuchen Sun, Shanhui Zhao, Tao Yu, Hao Wen, Samith Va, Mengwei Xu, Yuanchun Li, and Chongyang Zhang. Gui-xplore: Empowering generalizable gui agents with one exploration. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 19477–19486, June 2025.

[143] Yuxuan Sun, Manchen Wang, Shengyi Qian, William R Wong, Eric Gan, Pierluca D'Oro, Alejandro Castillejo Munoz, Sneha Silwal, Pedro Matias, Nitin Kamra, et al. Digidata: Training and evaluating general-purpose mobile control agents. arXiv preprint arXiv:2511.07413, 2025.

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

[145] Francesco Taioli et al. Mind the error! detection and localization of instruction errors in vision-and-language navigation. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2024.

[146] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. Advances in neural information processing systems, 30, 2017.

[147] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. Software Impacts, 6:100022, 2020.

[148] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. Transactions on Machine Learning Research, 2024.

[149] Han Wang, Archiki Prasad, Elias Stengel-Eskin, and Mohit Bansal. Retrieval-augmented generation with conflicting evidence. In Second Conference on Language Modeling, 2025.

[150] Jiangyuan Wang, Kejun Xiao, Qi Sun, Huaipeng Zhao, Tao Luo, Jian Dong Zhang, and Xiaoyi Zeng. Shoppingbench: A real-world intent-grounded shopping benchmark for llm-based agents. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 40, pages 33521–33529, 2026.

[151] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception, 2024.

[152] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. Frontiers of Computer Science, 18(6):186345, 2024.

[153] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers), pages 2609–2634, 2023.

[154] Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu.

Scienceworld: Is your agent smarter than a 5th grader? In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 11279–11298, 2022.

[155] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. arXiv preprint arXiv:1907.02057, 2019.

[156] Wenxuan Wang, Shi Juluan, Zixuan Ling, Yuk-Kit Chan, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, Wenxiang Jiao, and Michael R Lyu. Learning to ask: When llm agents meet unclear instruction. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 21784–21795, 2025.

[157] Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In International Conference on Learning Representations, volume 2024, pages 32593–32627, 2024.

[158] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In International Conference on Machine Learning, pages 63897–63911. PMLR, 2025.

[159] 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.

[160] Weiming Wu, Song-Lin Lv, Rui Zhu, Zi-Jian Cheng, and Lan-Zhe Guo. Can agents generalize to the open world? unveiling the fragility of static training in tool use. In Forty-third International Conference on Machine Learning, 2026.

[161] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. arXiv preprint arXiv:2309.07864, 2023.

[162] Junlin Xie, Zhihong Chen, Ruifei Zhang, Xiang Wan, and Guanbin Li. Large multimodal agents: A survey. arXiv preprint arXiv:2402.15116, 2024.

[163] Weimin Xiong, Yifan Song, Qingxiu Dong, Bingchan Zhao, Feifan Song, Xun Wang, and Sujian Li. Mpo: Boosting llm agents with meta plan optimization. arXiv preprint arXiv:2503.02682, 5(6):7, 2025.

[164] Tianshi Xu, Yuteng Chen, and Meng Li. Cleaner: Self-purified trajectories boost agentic reinforcement learning, 2026.

[165] Xilie Xu, Keyi Kong, Ning Liu, Lizhen Cui, Di Wang, Jingfeng Zhang, and Mohan Kankanhalli. An LLM can fool itself: A prompt-based adversarial attack. In The Twelfth International Conference on Learning Representations, 2024.

[166] Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. AndroidLab: Training and systematic benchmarking of android autonomous agents. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2144–2166, Vienna, Austria, July 2025. Association for Computational Linguistics.

[167] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. arXiv preprint arXiv:2310.11441, 2023.

[168] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024.

[169] Shunyu Yao, Howard Chen, John Yang, and Karthik R Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, Advances in Neural Information Processing Systems, 2022.

[170] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In The Eleventh International Conference on Learning Representations, 2023.

[171] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. In International Conference on Machine Learning, pages 11920–11931. PMLR, 2021.

[172] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 10674–10681, 2021.

[173] Wenhao Yuan, Chenchen Lin, Jian Chen, Jinfeng Xu, Xuehe Wang, and Edith Cheuk-Han Ngai. Verify before you commit: Towards faithful reasoning in llm agents via self-auditing. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 31201–31225, 2026.

[174] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In Findings of the Association for Computational Linguistics: ACL 2024, pages 3053–3077, 2024.

[175] Linda Zeng, Rithwik Gupta, Divij Motwani, Yi Zhang, and Diji Yang. Worse than zero-shot? a fact-checking dataset for evaluating the robustness of rag against misleading retrievals. Advances in Neural Information Processing Systems, 38, 2026.

[176] Weihao Zeng, Yuzhen Huang, and Junxian He. LOCA-bench: Benchmarking language agents under controllable and extreme context growth. In Forty-third International Conference on Machine Learning, 2026.

[177] Daochen Zha, Wenye Ma, Lei Yuan, Xia Hu, and Ji Liu. Rank the episodes: A simple approach for exploration in procedurally-generated environments. In International Conference on Learning Representations, 2021.

[178] Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Large language model-brained GUI agents: A survey. Transactions on Machine Learning Research, 2025.

[179] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, pages 1–20, 2025.

[180] Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 13643–13658, 2024.

[181] Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. Noveld: A simple yet effective exploration criterion. Advances in Neural Information Processing Systems, 34, 2021.

[182] Tuo Zhang, Alin-Ionut Popa, Yan Xu, Rui Song, and Dimitrios Dimitriadis. Pivot: Bridging planning and execution in llm agents via trajectory refinement, 2026.

[183] Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, et al. Toolbehonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 11388–11422, 2024.

[184] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao

Huang. Expel: Llm agents are experiential learners. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pages 19632–19642, 2024.

[185] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, Proceedings of the 41st International Conference on Machine Learning, volume 235 of Proceedings of Machine Learning Research, pages 61349–61385. PMLR, 21–27 Jul 2024.

[186] Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. Rtfm: Generalising to new environment dynamics via reading. In ICLR, pages 1–17. ICLR, 2020.

[187] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In Forty-first International Conference on Machine Learning, 2024.

[188] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In The Twelfth International Conference on Learning Representations, 2024.