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

Yao Fu[1] **Ran Qiu**[1] **Xinhe Wang**[1] **Jacob Sansom**[1] **Sathvika Ayyappa Prabhu**[1]
Huijie Tang[1] **Jaekyeom Kim**[2] **Sungryull Sohn**[2] **Honglak Lee**[1,2]
[1]University of Michigan [2]LG AI Research

Abstract

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 make up for limited domain-specific knowledge. Such guidance includes human-provided manuals and demonstrations, retrieved examples from memory or external tools, 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environment, ambiguous or simplified language, or retrieval errors from external sources, guidance can be incomplete, outdated, or contextually mismatched, potentially causing errors or failures during task execution. To address this, we introduce **MIRAGE**, a benchmark for ***M**easur**I**ng **R**obustness of LLM **A**gents under Imperfect **G**uidanc**E***. 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 using WebArena, incorporating noisy or underspecified instructions derived from human demonstrations. Our findings reveal critical failure modes in current LLM agents and motivate future work on improving their robustness under imperfect guidance.

1 Introduction

Recent advances in large language models (LLMs) have made them increasingly reliable at solving complex decision-making tasks in diverse interactive environments (Xi et al., 2023; Yao et al., 2022b; Liu et al., 2023), including web navigation (Zhang et al., 2025), embodied control (Liang et al., 2023), and open-ended games (Paglieri et al., 2025; Wang et al., 2023a). 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) (Kagaya et al., 2024), reusable workflows (Wang et al., 2024b), natural language cues and strategies acquired through prior interactions and memory (Zhao et al., 2023; Fu et al., 2024), or clarifications and hints from interactive dialogue with users (Wang et al., 2025; Murzaku et al., 2025).

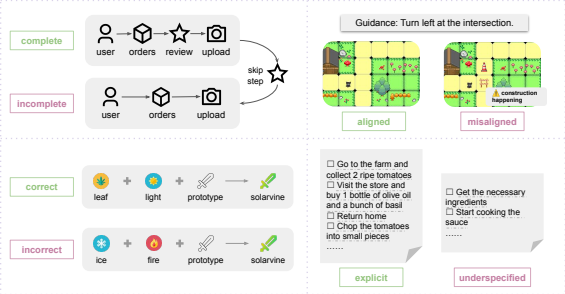

Figure 1: Examples of four types of guidance imperfection. (1) Incomplete – a critical step (e.g., uploading a photo to review an order) is omitted; (2) Incorrect – a weapon crafting recipe includes wrong components; (3) Misaligned – originally correct navigation guidance ("turn left") becomes invalid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

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, oversimplifications, or subtle misstatements of relevant information (Huang et al., 2025). Second, guidance may be misaligned with the environment, especially in dynamic, noisy settings (Pan et al., 2024). A plan that was once valid

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 6591–6618

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** (**M**easur**I**ng **R**obustness of LLM **A**gents under Imperfect **G**uidanc**E**), a two-part benchmark suite comprising: (1) MIRAGE-World, a collection of procedurally generated environments spanning navigation, cooking, and gaming tasks; and (2) MIRAGE-Web, a set of 450 realistic web-based tasks derived from the WebArena platform (Zhou et al., 2024b) 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 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. 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.

2 Related Work

Instruction Following in LLMs. As LLMs become better at following instructions (He et al., 2024; Heo et al., 2025), guidance retrieval has emerged as a promising approach for improving agent performance (Zhao et al., 2023; Chen et al., 2024), though its success still highly depends on the quality of the guidance. In practice, guidance is often incomplete, ambiguous, or misleading (Paxton et al., 2019; Roh et al., 2020; Su et al., 2024). This limitation is particularly well-documented in domains such as embodied navigation (Lin et al., 2021), robotics (Paxton et al., 2019), and autonomous driving (Roh et al., 2020), 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 (Wang et al., 2025; Qian et al., 2024; Andukuri et al., 2024), showing that LLMs often make incorrect assumptions to compensate for ambiguity, and on robustness against adversarial attacks (Paulus et al., 2024; Kumar et al., 2023; Xu et al., 2023), 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 (Shinn et al., 2023; Fu et al., 2025; Zhou et al., 2024a) aim to improve performance by iteratively revising plans or prompting models to reason about their past failures. While methods such as Reflexion (Shinn et al., 2023) incorporate explicit mechanisms for self-reflection, recent findings suggest that these mechanisms are often brittle

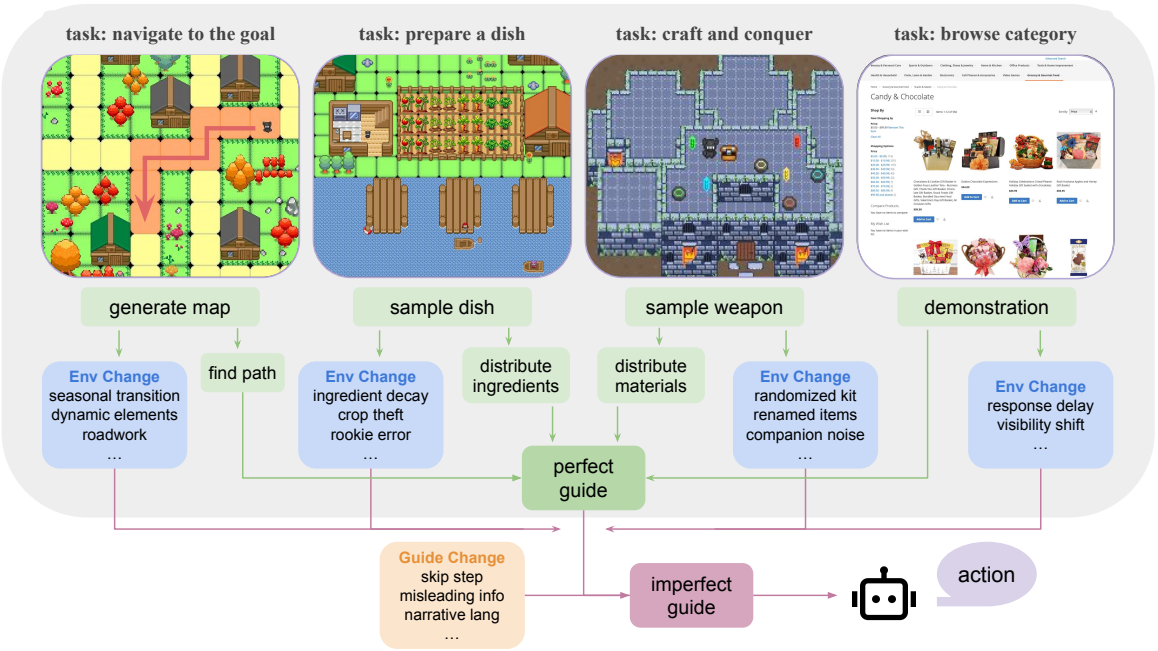

Figure 2: Overview of MIRAGE. For navigation, cooking, and gaming, we first generate the environment and task instance (e.g., map layout, recipe goal, or target boss). From this, we derive a perfect solution path, which is then optionally perturbed to produce an imperfect guide. For web tasks, we extract 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 .

and unreliable, particularly in environments with noisy or misleading feedback (Shinn et al., 2023; Wang et al., 2023a; Kambhampati et al., 2024). 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 (Liu et al., 2023; Wang et al., 2022; Shridhar et al., 2021; Yao et al., 2022a; Paglieri et al., 2025; Zhou et al., 2024b). However, these environments do not specifically test LLM agents' robustness under imperfect guidance. MINT (Wang et al., 2024a) 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 (Ma et al., 2022a) 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.

2.1 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 guidance `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.

2.2 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 (Zhou et al., 2023). As shown in Figure 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 1 gives a domain-level overview. Additional environment details are provided in Appendix B.

2.2.1 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 differs in layout across episodes. Agents receive a 3x3 partial 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").

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 (Zhou et al., 2023), 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.

2.3 Guidance Imperfection

MIRAGE provides a unified framework for modeling and evaluating imperfect guidance across diverse domains. As illustrated in Figure 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

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

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 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.

2.4 Environment Implementation and Task Construction

All environments in MIRAGE-World are powered by a fully custom simulation engine built from scratch, using Pygame (Pygame, 2024) and CUTERPG pack (PixyMoon, n.d.). 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 (Brockman et al., 2016) for agent interaction and a graphical interface for human use and debugging. For MIRAGE-Web, we extend the WebArena framework (Zhou et al., 2024b) 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.

2.5 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

Task	Guidance	Human	ReAct					P&S	Reflexion
			GPT-4.1 mini	GPT-4o mini	Qwen-3 14B	Llama-3 70B	Qwen-3 235B	GPT-4.1 mini	GPT-4.1 mini
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 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.

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 Experiments

We evaluate a mix of LLM models and agent frameworks. In `MIRAGE-World`, we benchmark five models—GPT-4.1-mini (Kumar et al., 2025), GPT-4o-mini (Menick et al., 2024), Llama-3-70B (Grattafiori et al., 2024), Qwen-3-14B (Qwen Team, 2025), and Qwen-3-235B (Qwen Team, 2025), 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 D.

3.1 MIRAGE-World Results

We evaluate agent performance under imperfect guidance using three representative baseline methods. (1) ReAct (Yao et al., 2022b) 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 (Wang et al., 2023b; Jansen et al., 2024). The agent incrementally generates and executes one plan step at a time with a separate ReAct agent. (3) Reflexion (Shinn et al., 2023) 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 2 reports task success rates under varying guidance conditions. We summarize several key findings:

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 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 lies in long-horizon reasoning, but in re-

[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.

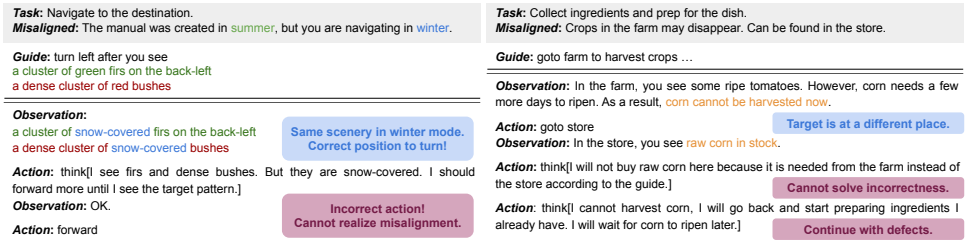

Figure 3: Two examples where the agent fails under imperfect guidance. Left: it does not recognize the misalignment. Right: it realizes the guidance is inapplicable but cannot recover and continues with defects.

Domain	Perfect	Incomplete	Underspecific		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: ReAct success rate with GPT-4o on WebArena tasks with imperfect guidance.

covering 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 benefits 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 pretraining knowledge. These findings validate the design of MIRAGE-World: it presents diverse yet solvable challenges that probe the guidance-following behavior of LLM agents.

3.2 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. 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.

3.3 Additional Analyses

3.3.1 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 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**	**Corret Obs**	**Seasonal Obs**	**Perfect**	**Storage Loss**
NA	90%	22%	8%	46%	33%	65%	48%
IA	90%	16%	6%	43%	30%	58%	60%

Table 4: Comparison between **NA** = Non-Aware and **IA** = Imperfect-Aware agents.

3.3.2 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 omitting increasing proportions of steps from the demonstration steps: 0% (perfect), 25%, 50%, and 100% of steps skipped.

As shown in Figure 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.

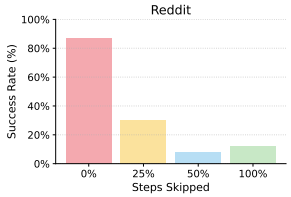

Figure 4: Effect of skipping steps in web guidance.

3.3.3 Error Analysis

Figure 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.

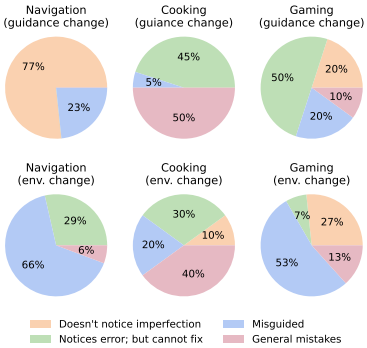

Figure 5: Error types of GPT-4.1-mini ReAct agents.

4 Conclusion

In this paper, 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 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.

Limitations

While MIRAGE offers a flexible testbed for evaluating agent robustness under imperfect guidance, several limitations remain. First, our experiments focus entirely on prompted language models that are fixed and pretrained. We do not explore how agents behave when fine-tuned using guidance-augmented data. This leaves several important

questions unanswered. For example, does training only on perfect guidance make a model more fragile when later exposed to flawed instructions? If trained on a subset of imperfect cases, can a model generalize to other types of imperfection it has not seen? What forms of reasoning traces are most helpful for improving robustness during training? Since MIRAGE-World supports large-scale procedural generation of tasks and guidance variations, it provides a strong foundation for conducting controlled finetuning experiments. Understanding how the type, frequency, and severity of imperfection, along with the structure of reasoning traces provided during training, influence generalization at test time remains an important direction for future research. Second, although MIRAGE-Web includes realistic simulated changes such as renamed interface elements, modified workflows with missing steps, and delayed observations, it does not fully capture the long-term structural changes that occur on real websites. In practice, websites evolve over time through layout changes across versions, the removal or replacement of features, and variation in response times. While our simulations are designed to reflect plausible and realistic shifts, incorporating real-world data, such as archived snapshots of widely used websites, would support more realistic and faithful benchmarking in dynamic digital environments.

References

Chinmaya Andukuri, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah D. Goodman. 2024. Star-gate: Teaching language models to ask clarifying questions. *Preprint*, arXiv:2403.19154. Preprint.

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

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. 2024. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. *Preprint*, arXiv:2405.16247.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma Gongque, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. 2025. Agentrefine: Enhancing agent generalization through refinement tuning. *Preprint*, arXiv:2501.01702.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Preprint*, arXiv:2403.08978.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783. Preprint.

Qianyu He, Jie Zeng, Qianxi He, Jiaqing Liang, and Yanghua Xiao. 2024. From complex to simple: Enhancing multi-constraint complex instruction following ability of large language models. *Preprint*, arXiv:2404.15846.

Juyeon Heo, Christina Heinze-Deml, Oussama Elachqar, Kwan Ho Ryan Chan, Shirley Ren, Udhay Nallasamy, Andy Miller, and Jaya Narain. 2025. Do llms "know" internally when they follow instructions? *Preprint*, arXiv:2410.14516.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55.

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. 2024. Discoveryworld: A virtual environment for developing and evaluating automated scientific discovery agents. *Preprint*, arXiv:2406.06769.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. 2024. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents. *Preprint*, arXiv:2402.03610.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. 2024. Llms can't plan, but can help planning in llm-modulo frameworks. *Preprint*, arXiv:2402.01817.

Ananya Kumar, Jiahui Yu, John Hallman, Michelle Pokrass, and 1 others. 2025. Introducing gpt-4.1 in the api.

Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Aaron Jiaxun Li, Soheil Feizi, and Himabindu Lakkaraju. 2023. Certifying llm safety against adversarial prompting. *Preprint*, arXiv:2309.02705. Preprint.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. *Preprint*, arXiv:2209.07753.

Bingqian Lin, Yi Zhu, Yanxin Long, Xiaodan Liang, Qixiang Ye, and Liang Lin. 2021. Adversarial reinforced instruction attacker for robust vision-language navigation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):7175–7189.

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, and 3 others. 2023. Agentbench: Evaluating llms as agents. *Preprint*, arXiv:2308.03688.

Ziqiao Ma, Ben VanDerPloeg, Cristian-Paul Bara, Huang Yidong, Eui-In Kim, Felix Gervits, Matthew Marge, and Joyce Chai. 2022a. Dorothie: Spoken dialogue for handling unexpected situations in interactive autonomous driving agents. *Preprint*, arXiv:2210.12511.

Ziqiao Ma and 1 others. 2022b. Dorothie: Spoken dialogue for handling unexpected situations in interactive autonomous driving agents. *Preprint*, arXiv:2210.12511. Preprint.

Jacob Menick, Kevin Lu, Shengjia Zhao, Eric Wallace, Hongyu Ren, Haitang Hu, Nick Stathas, and Felipe Petroski Such. 2024. Gpt-4o mini: advancing cost-efficient intelligence.

John Murzaku, Zifan Liu, Md Mehrab Tanjim, Vaishnavi Muppala, Xiang Chen, and Yunyao Li. 2025. Eclair: Enhanced clarification for interactive responses. *Preprint*, arXiv:2503.15739.

OpenAI. 2024. Hello GPT-4. https://openai.com/index/hello-gpt-4o/. Accessed: 2024-05-16.

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. 2025. Balrog: Benchmarking agentic llm and vlm reasoning on games. *Preprint*, arXiv:2411.13543.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. 2024. Webcanvas: Benchmarking web agents in online environments. *Preprint*, arXiv:2406.12373.

Anselm Paulus, Arman Zharmagambetov, Chuan Guo, Brandon Amos, and Yuandong Tian. 2024. Advprompter: Fast adaptive adversarial prompting for llms. *Preprint*, arXiv:2404.16873. Preprint.

Chris Paxton, Yonatan Bisk, Jesse Thomason, Arunkumar Byravan, and Dieter Foxl. 2019. Prospection: Interpretable plans from language by predicting the future. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6942–6948. IEEE.

PixyMoon. n.d. CUTERPG pack. https://pixymoon.itch.io/cuterpg. Accessed: 2025-06-23.

Pygame. 2024. Pygame. https://www.pygame.org/. Accessed: 2024-06-05.

Cheng Qian, Bingxiang He, Zhong Zhuang, Jia Deng, Yujia Qin, Xin Cong, Zhong Zhang, Jie Zhou, Yankai Lin, Zhiyuan Liu, and 1 others. 2024. Tell me more! towards implicit user intention understanding of language model-driven agents. *Preprint*, arXiv:2402.09205. Preprint.

Qwen Team. 2025. Qwen3: Think deeper, act faster.

Junha Roh, Chris Paxton, Andrzej Pronobis, Ali Farhadi, and Dieter Fox. 2020. Conditional driving from natural language instructions. In *Conference on Robot Learning*, pages 540–551. PMLR.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *ICLR*.

Zhe Su, Xuhui Zhou, Sanketh Rangreji, Anubha Kabra, Julia Mendelsohn, Faeze Brahman, and Maarten Sap. 2024. Ai-liedar: Examine the trade-off between utility and truthfulness in llm agents. *Preprint*, arXiv:2409.09013. Preprint.

Alane Suhr and 1 others. 2019. Executing instructions in situated collaborative interactions. *Preprint*, arXiv:1910.03655. Preprint.

Francesco Taioli and 1 others. 2024. 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.

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

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023b. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *Preprint*, arXiv:2305.04091.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? *Preprint*, arXiv:2203.07540.

Wenxuan Wang, Juluan Shi, Zixuan Ling, Yuk-Kit Chan, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen tse Huang, Wenxiang Jiao, and Michael R. Lyu. 2025. Learning to ask: When llm agents meet unclear instruction. *Preprint*, arXiv:2409.00557.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024a. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *Preprint*, arXiv:2309.10691.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024b. Agent workflow memory. *Preprint*, arXiv:2409.07429.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, and 10 others. 2023. The rise and potential of large language model based agents: A survey. *Preprint*, arXiv:2309.07864.

Xilie Xu, Keyi Kong, Ning Liu, Lizhen Cui, Di Wang, Jingfeng Zhang, and Mohan Kankanhalli. 2023. An llm can fool itself: A prompt-based adversarial attack. *Preprint*, arXiv:2310.13345. Preprint.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022a. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022b. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629. Preprint.

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. 2025. Large language model-brained gui agents: A survey. *Preprint*, arXiv:2411.18279.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners. *Preprint*, arXiv:2308.10144. Preprint.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024a. Language agent tree search unifies reasoning acting and planning in language models. *Preprint*, arXiv:2310.04406.

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

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. 2024b. Webarena: A realistic web environment for building autonomous agents. *Preprint*, arXiv:2307.13854.

A 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 work contributes to the advancement of agents that are not only capable but also trustworthy and adaptable.

B 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 (Zhou et al., 2023) 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. We plan to release all code upon acceptance of the paper. 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 work will be properly cited and linked in the release documentation.

C MIRAGE Benchmark Framework

C.1 MIRAGE-World Navigation

C.1.1 Task Overview

In this task, the agent must navigate a procedurally generated grid environment to reach a destination. All experiments presented in the paper 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 time step, 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.

Benchmark	Domain(s)	Task Guidance	Guidance Change?	Env Change?	Easy Extensibility	Procedurally Generated?	Agent Autonomy
WebArena (Zhou et al., 2023)	web	✗	✗	✗	✗	✗	Single-Agent
CEREAL-BAR (Suhr et al., 2019)	game	✓	✗	✗	✗	✗	Dialogue
R2RIE-CE (Taioli et al., 2024)	navigation	✓	✓ (incorrect)	✗	✗	✗	Single-Agent
SDN (Ma et al., 2022b)	autonomous driving	✓	✗	✓	✗	✗	Dialogue
MIRAGE (ours)	navigation, cook, game, web	✓	✓ (incorrect, underspecified, incomplete)	✓	✓	✓ (for Mirage-World)	Single-Agent

Table 5: Comparison of MIRAGE and related benchmarks.

- `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

$$\text{reward} = 1 - \frac{\text{Distance to destination}}{\text{Maximum possible distance}} \quad (1)$$

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.

C.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.

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 6.

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

Figure 6: 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.

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.

C.2 MIRAGE-World Cooking

C.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.

C.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.

C.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.

C.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.

C.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:

$$\text{reward} = \frac{\text{\# of qualified dishes served}}{\text{\# of required dishes}} \quad (2)$$

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.

C.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.

C.3 MIRAGE-World Gaming

C.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.

C.3.2 Task Generation

Each roguelike episode is procedurally generated based on a predefined level layout. In the standard configuration used throughout this paper, 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.

C.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).

C.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.

C.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:

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

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

C.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. However, the environment supports a `recipes` function that the agent can invoke at any time to retrieve the true, valid 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".

C.4 MIRAGE-Web WebArena

C.4.1 Overview

WebArena (Zhou et al., 2023) 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.

C.4.2 Observation Space

At each time step, 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.

C.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.

C.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, and therefore recruitment and payment are not applicable. 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.

C.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.

D 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 (Yao et al., 2022b), 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 (Wang et al., 2023b; Jansen et al., 2024). 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 6: 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 (Shinn et al., 2023) 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.

D.1 Hyper-parameters

For all MIRAGE-World experiments, we use the following language models: GPT-4.1-mini (Kumar et al., 2025), GPT-4o-mini (Menick et al., 2024), Llama-3-70B (Grattafiori et al., 2024), Qwen-3-14B (Qwen Team, 2025), and Qwen-3-235B (Qwen Team, 2025). 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 (OpenAI, 2024), as the tasks involve more realistic and challenging web tasks with long, complex observations that demand stronger contextual reasoning.

D.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.

D.3 Results

D.3.1 MIRAGE-World Results

We present the full results of ReAct-style agents across different models in Table 6. 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 7. 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.

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

D.3.2 MIRAGE-Web Results

Results for MIRAGE-Web are shown in Table 9, evaluated using GPT-4o with a ReAct-style agent across different types of guidance imperfections. We consider the following variants:

- **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.

D.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 10 shows that ambiguity harms ReAct, but is easy to identify and recover via imperfect-aware prompting.

E Prompts

E.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 8, Figure 9, and Figure 7 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.

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 8: 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	Underspecific		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 9: 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.

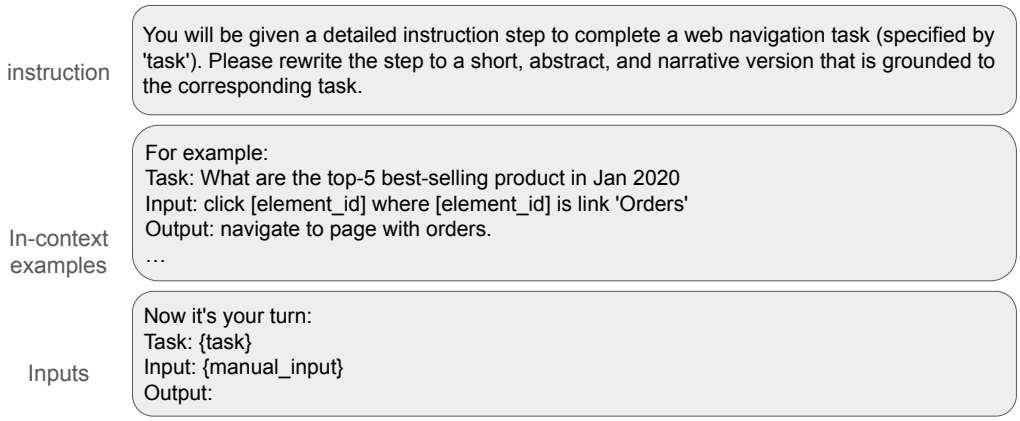

Figure 7: Example of narrative-style rewrite. Guidance are rewritten in a more descriptive form.

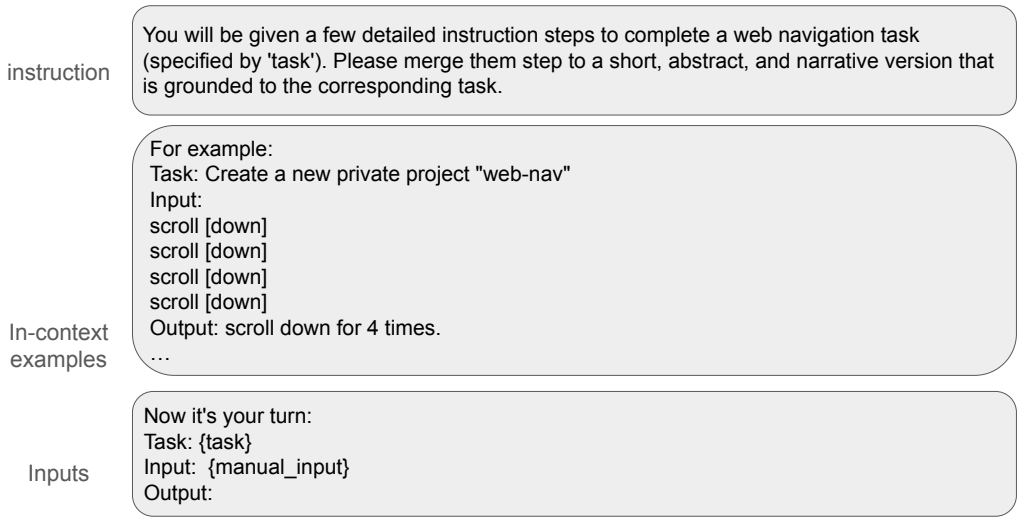

Figure 8: Example of merged rewrite. Multiple consecutive steps are compressed into a single abstract description.

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

Table 10: 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.

F Use of AI Writing Assistants

We used large language models (e.g., ChatGPT) for minor language editing and phrasing improvements in the writing of this paper. The use of these tools was limited to stylistic refinement; all research design, implementation, analysis, and conclusions were entirely conducted and authored by ourselves.

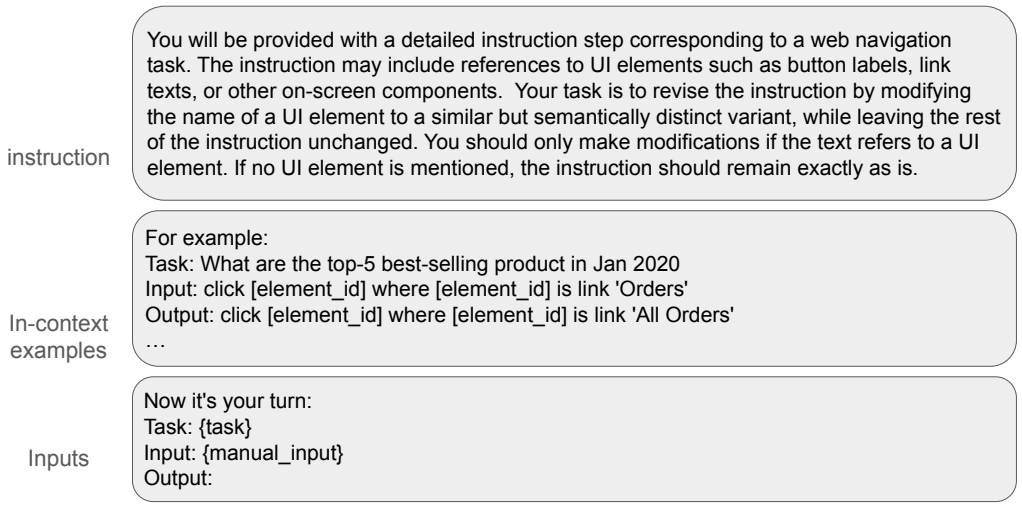

Figure 9: Example of UI label variation. Labels for interface elements are changed to introduce ambiguity or misalignment.