Agent-Environment Alignment via Automated Interface Generation

Kaiming Liu[1,2], Xuanyu Lei[2,3], Ziyue Wang[2,3], Peng Li[2,†], Yang Liu[1,2,3,†]
[1]College of AI, Tsinghua University, Beijing, China
[2]Institute for AI Industry Research (AIR), Tsinghua University, Beijing, China
[3]Dept. of Comp. Sci. & Tech., Institute for AI, Tsinghua University, Beijing, China
lkm20@mails.tsinghua.edu.cn, lipeng@air.tsinghua.edu.cn
liuyang2011@tsinghua.edu.cn

Abstract

Large language model agents have shown impressive capabilities in interactive decision-making tasks. These agents interact with environment through intermediate interfaces, such as predefined action spaces and interaction rules, which mediate the perception and action. However, mismatches often happen between the internal expectations of the agent regarding the influence of its issued actions and the actual state transitions in the environment, a phenomenon referred to as **agent-environment misalignment**. While prior work has invested substantially in improving agent strategies and environment design, the critical role of the interface still remains underexplored. In this work, we empirically demonstrate that agent-environment misalignment poses a significant bottleneck to agent performance. To mitigate this issue, we propose **ALIGN**, an Auto-Aligned Interface Generation framework that alleviates the misalignment by enriching the interface. Implemented as a lightweight wrapper, ALIGN-generated interface achieves the alignment without modifying either the agent logic or the environment code. Experiments across multiple domains including embodied tasks, web navigation and tool-use, achieve consistent performance improvements, with a 45.67% success rate improvement in ALFWorld.

1 Introduction

Large Language Model (LLM) agents have demonstrated promising performance in interactive tasks such as embodied tasks (Wang et al., 2024a), web navigation tasks (Qi et al., 2024), and tool-use tasks (Schick et al., 2023). In these tasks, agents typically interact with the environment through manually designed **interfaces** such as predefined action spaces and interaction rules. While substantial efforts have been devoted to improving agents and environments, comparatively little attention has been paid to the interface between them, leading to a problem we term **agent-environment misalignment**, which impacts the agent performance.

The agent-environment misalignment refers to the discrepancy between the interpretation of the agent to the observation following an action and the underlying logic of the environment. As illustrated in Figure 1 (left), in ALFWorld (Shridhar et al., 2021), an invalid action returns "Nothing happens", which agents misinterpret as an empty state rather than a constraint violation. This misalignment significantly hinders task success and imposes unnecessary cognitive overhead on agents. From the perspective of the agent, poorly designed interfaces impose unnecessary cognitive overhead. Furthermore, from an evaluation perspective, inadequate interfaces can obscure an accurate assessment of the true reasoning capabilities of agents. Therefore, we argue that the problem of agent-environment misalignment warrants greater attention.

However, addressing the agent-environment misalignment is challenging. On one hand, current benchmarks primarily focus on advance agent intelligence by constructing increasingly complex and challenging environments (Xie et al., 2024; Zhou et al., 2024a), often overlooking the importance of improving interface design. This oversight extends across multiple domains of interactive tasks, such as, ALFWorld, OSWorld (Xie et al., 2024), and M^{3}ToolEval (Wang et al., 2024b). They all exhibit similar deficiencies: failing to provide agent-parseable observations for environmental constraints violation in embodied tasks, positional inaccuracies in operating system tasks or parameter format errors in multi-turn tool-use tasks, respectively. On the other hand, although some recent work (Yang et al., 2024b; Zheng et al., 2024) has begun to consider interface design, these efforts

† Corresponding Authors.
‡ Code and data are available at https://github.com/THUNLP-MT/ALIGN.

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 1480–1512
November 6-10, 2026 ©2026 Association for Computational Linguistics

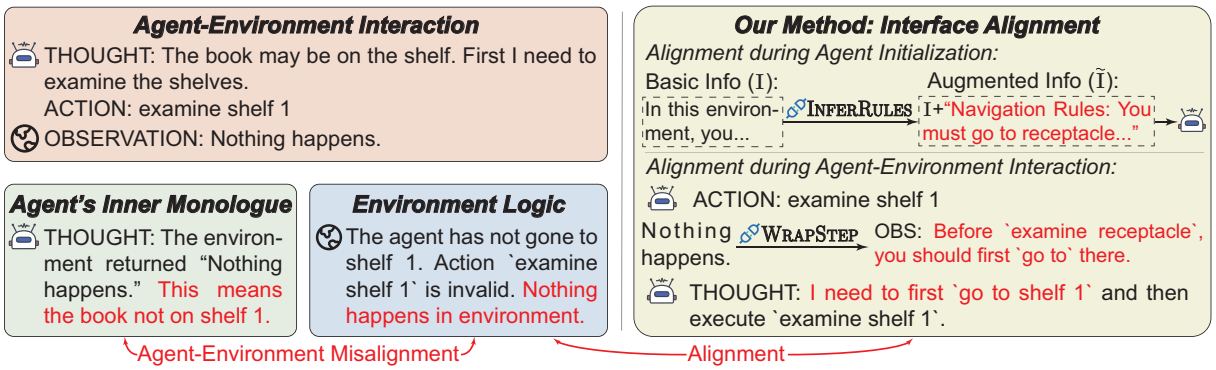


Figure 1: **Illustration of agent-environment misalignment and our proposed solution. Left:** The agent and the environment have a misalignment in their interpretation of the same observation, where the agent's understanding of the observation differs from the environment's underlying logic. **Right:** ALIGN bridges this gap via an automatically generated interface comprising two modules, **InferRules** and **WrapStep**: (1) During initialization, InferRules augments the basic information with explicit environment constraints. (2) During interaction, WrapStep translates the raw observation into an informative observation.

often rely on manual, environment-specific tailoring, which introduces two critical issues: (1) they are highly labor-intensive and (2) whether human-designed interfaces are optimal for agents remains an open question. Furthermore, researchers often manually re-engineer interfaces for specific methods (Ma et al., 2024; Chen et al., 2024a), compromising comparability and making it unclear whether improvements stem from novel architectures or customized interfaces.

Distinct from the aforementioned works, we propose to **automatically generate interfaces for bridging the agent-environment misalignment**. In this work, we introduce **ALIGN** (Auto-Aligned Interface Generation), a framework that automatically generate aligned interfaces for environments. The generated interface consists of two modules: InferRules and WrapStep. The former discovers and provides the agent with static information about environmental rules or internal constraints, facilitating *static alignment*, while the latter enhances the interaction by offering more detailed observations for agent-issuing actions, enabling *dynamic alignment*, as shown in Figure 1 (right). The generated interface acts as a lightweight wrapper, providing richer context and explicit constraint hints, enabling different LLM agents to align with the environment directly.

To evaluate the effectiveness of ALIGN, we conduct experiments on four representative benchmarks across three domains: embodied tasks, web navigation, and tool-use tasks. Our results demonstrate consistent performance improvements across all four benchmarks when using the ALIGN-generated interface, with notably gains of 45.67% in average success rate on ALFWorld. Moreover, the performance of GPT-4.1-based agents on ALFWorld are improved from 73.88% to 93.28% with ALIGN, highlighting the efficiency of our approach in mitigating the agent-environment misalignment to unleash the agent's true capabilities.

Our contributions are summarized as follows:

- We identify and characterize the **agent-environment misalignment**, empirically showing its prevalence across diverse domains and its role as a key bottleneck to performance.
- We introduce **ALIGN**, the first framework that automatically generates aligned interfaces to alleviate agent-environment misalignment, without modifying agent logic or environment code.
- We demonstrate the effectiveness and generalizability of **ALIGN** across three domains, with up to a 45.67% success rate gain on ALFWorld.

2 Related Work

Agent-environment interface The agent-environment interface defines how agents interact with the environment. In reinforcement learning, unified interaction interfaces (Bonnet et al., 2024; Brockman et al., 2016; Kolve et al., 2017; Towers et al., 2024) standardize the evaluation of different algorithms. As LLMs become increasingly capable of human-like actions (Guo et al., 2024; Liu et al., 2024; Ma et al., 2024), interface design has been shown to largely influence LLM-based agent performance (Xie et al., 2024; Rawles et al., 2024). SWE-agent (Yang et al., 2024b) proposes agent-

computer interfaces for coding agents, and recent efforts aim to improve generalization (Agashe et al., 2024; Qin et al., 2025; Niu et al., 2024) or augment interfaces with auxiliary tools (Bula et al., 2025; Gou et al., 2024; Lei et al., 2025; Lu et al., 2024; Yang et al., 2023a). However, these interfaces are mostly hand-crafted for specific environments or frameworks, limiting their generalization and scalability. We instead propose automated interface generation for effective, generalizable alignment.

Methods aligning agents with environments LLM agents have shown strong potential for real-world interaction and task completion (Yao et al., 2023; Shinn et al., 2023; Liu et al., 2024), with existing work falling into training-based and training-free categories. Training-based methods fine-tune LLMs on expert trajectories (Zeng et al., 2024; Chen et al., 2023, 2025; Fu et al., 2025; Chen et al., 2024b) or apply reinforcement learning for environment-aligned planning (Bai et al., 2025; Yang et al., 2024c; Qi et al., 2024; Feng et al., 2024; Zhou et al., 2024c; Wang et al., 2025), but suffer from high computational costs and poor generalization to unseen environments. Training-free methods build multi-agent frameworks for task decomposition and experience accumulation (Chen et al., 2024a; He et al., 2024; Sun et al., 2024; Yang et al., 2023b; Zhou et al., 2024b), yet their static pipelines and prompt-injected experience often fail to capture environment dynamics, yielding insufficient agent-environment alignment.

Rule learning and prompt optimization Closest to our work are methods that learn *agent-side* knowledge: WALL-E (Zhou et al., 2024b) induces rules that correct the agent's world model, AutoManual (Chen et al., 2024a) builds a manual that conditions its planner, and automatic prompt optimization (APO) methods (Zhou et al., 2023; Pryzant et al., 2023; Yang et al., 2024a; Wang et al., 2024c) refine the agent's prompt with an LLM-driven search loop. ALIGN instead optimizes the environment-side functions f_{info} and f_{obs}, produces executable interface code rather than natural-language knowledge, and accepts a change only after controlled experiments in the environment. Since these methods leave the feedback channel untouched, they consume exactly the raw observations that misalignment corrupts and are thus complementary to ALIGN (Section 4.6); a detailed comparison is given in Appendix D.1.

3 Method

3.1 Problem Formulation

Environment and Agent. In interactive decision tasks, we define the environment $\mathcal{E}$ as a tuple $(\mathcal{S}, \mathcal{A}, T, F, \mathcal{I})$, where $\mathcal{S}$ denotes the set of all possible states of the environment; $\mathcal{A}$ denotes the set of actions the agent can invoke; $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ defines how the environment state evolves in response to agent actions; $F : \mathcal{S} \times \mathcal{A} \to \mathcal{O}$ provides textual feedback that reflects the consequences of the action in the current state, where $\mathcal{O}$ is all possible observations; $\mathcal{I}$ encodes the *environment foundational information description*, a fixed, declarative representation of the environment's basic introduction, which is exposed to the agent at initialization. An agent π interacts with environment $\mathcal{E}$ by receiving task description and observations, then producing actions $a_t \in \mathcal{A}$. The interaction generates trajectory $\tau = [(s_0, a_0, o_0), \ldots, (s_T, a_T, o_T)]$, culminating in task completion feedback.

Formal Definition of Interface. Existing works typically assume the agent interacts directly with $\mathcal{E}$. However, we argue that this interaction is mediated by an **Interface**, denoted as Φ, which acts as a translation layer between the agent's cognitive space and the environment's execution space. Formally, we define the interface as a tuple of mapping functions: $\Phi = \langle f_{\text{info}}, f_{\text{act}}, f_{\text{obs}} \rangle$where each component serves a distinct role:

- **Information Augmenter** $f_{\text{info}} : \mathcal{I} \to \tilde{\mathcal{I}}$ exposes implicit environment logic (e.g., admissible action sequences) into an explicit descriptive context $\tilde{\mathcal{I}}$ provided at agent initialization.
- **Action Transducer** $f_{\text{act}} : \mathcal{A}_{\text{agent}} \to \mathcal{A}_{\text{env}} \cup \{\perp\}$ maps the agent's output to an executable environment command. If the output cannot be parsed, it returns an invalid symbol $\perp$.
- **Observation Transducer** $f_{\text{obs}} : \mathcal{S} \times \mathcal{A}_{\text{env}} \times \mathcal{O}_{\text{raw}} \to \mathcal{O}_{\text{agent}}$ transforms the raw feedback $\mathcal{O}_{\text{raw}}$ (from F) into an informative observation $\mathcal{O}_{\text{agent}}$ that better conveys the actual state transitions and their causes.

At each timestep t, the agent receives $\tilde{o}_t \in \mathcal{O}_{\text{agent}}$ (processed by f_{obs}) and generates $a_t \in \mathcal{A}_{\text{agent}}$, which is then executed as $a_t^{\text{env}} = f_{\text{act}}(a_t) \in \mathcal{A}_{\text{env}}$.

Agent-Environment Misalignment. We analyze the misalignment problem through the lens of the interface Φ. Ideally, Φ should be *lossless*, maximizing the mutual information between the agent's belief state and the ground-truth environment state.

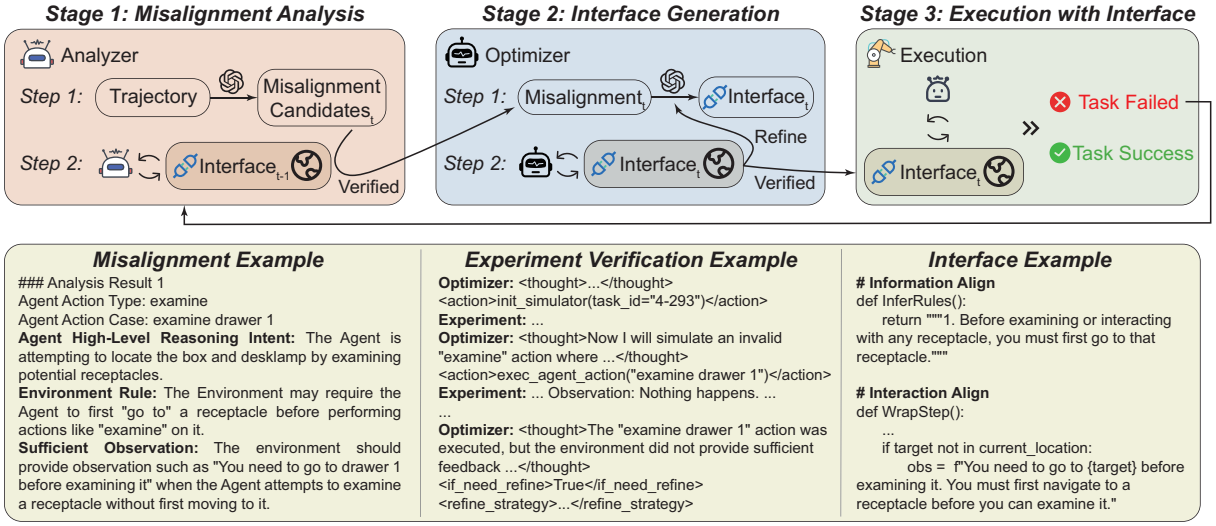


Figure 2: **ALIGN framework.** In each iteration, ALIGN progresses though three stages. **Stage 1**: the Analyzer identifies potential agent-environment misalignments and validates them through experiments; **Stage 2**: the Optimizer generates a new interface based on the previous interface and identified misalignments, followed by verification and refinement; **Stage 3**: the agent interacts with the updated interface-wrapped environment, with trajectories of failed tasks fed back to the Analyzer for analysis in the next iteration. At the bottom of the figure, examples for misalignment, verification of interface by Optimizer, and the ALIGN-generated interface are provided.

However, manually designed interfaces often exhibit **Semantic Gaps** through two mechanisms: (1) *state aliasing*, where a lossy f_{obs} maps distinct error states (e.g., wrong location vs. missing object) to the same generic observation (e.g., "Nothing happens."), preventing the agent from diagnosing failures; and (2) *under-specified constraints*, where critical preconditions of T (e.g., "open" requires "`go to`" first) are not encoded in $\tilde{\mathcal{I}}$ by f_{info}, leaving the agent under a hallucinated world model.

Therefore, we define **Agent-Environment Misalignment** as the discrepancy between the agent's expected state transition $s_{t+1}^{\text{expected}}$ (derived from its internal world model based on $\tilde{\mathcal{I}}$ and prior observations $[\tilde{o_0}, \tilde{o_1}, \ldots, \tilde{o_{t+1}}]$) and the actual transition $s_{t+1}^{\text{actual}} = T(s_t, a_t^{\text{env}})$, caused by insufficient expressiveness of the interface Φ.

Scope of This Work. We observe that misalignment in existing benchmarks primarily stems from information loss in f_{info} and f_{obs} rather than action space incompatibility. ALIGN therefore treats f_{act} as the identity mapping and focuses on automatically optimizing the other two components. An audit of all 87 `env.step` call sites in the 20 generated interfaces supports this choice empirically: none remaps an agent action to a semantically different one (Appendix D.6).

3.2 ALIGN Overview

We introduce **ALIGN** (Auto-Aligned Interface Generation), a framework that automatically generates an optimized interface Φ^* to bridge the semantic gaps identified in Section 3.1. As illustrated in Figure 3, ALIGN instantiates f_{info} and f_{obs} as two learnable modules implemented as a lightweight Python wrapper:

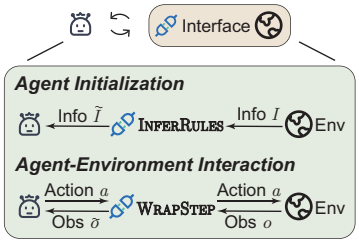


Figure 3: Overview of the ALIGN-generated interface. InferRules enriches static information ($\mathcal{I} \to \tilde{\mathcal{I}}$) at agent initialization, while WrapStep augments stepwise observations ($F \to \tilde{F}$) during interaction.

Interface Instantiation. As illustrated in Figure 3, ALIGN instantiates the theoretical interface components through two learnable modules implemented as a lightweight Python wrapper, without modifying the environment code or agent logic.

InferRules (implements f_{info}) extracts interaction rules and constraints from $(\text{task}, o_0, \mathcal{I})$ to produce $\tilde{\mathcal{I}}$, exposing precondition dependencies and action ordering requirements at initialization.

WRAPSTEP (implements f_{obs}) intercepts F and augments each raw observation with diagnostic information to resolve state aliasing in interaction.

Together, these modules wrap $\mathcal{E}$ into an *augmented environment* $\tilde{\mathcal{E}} = (\mathcal{S}, \mathcal{A}, T, \tilde{F}, \mathcal{I} \cup \tilde{\mathcal{I}})$ with $\tilde{F}(s_t, a_t) := \text{WRAPSTEP}(F, s_t, a_t)$, leaving the agent logic, the environment code, and the transition dynamics T unchanged. We denote the interface as $\Phi := \{\text{INFERRULES}, \text{WRAPSTEP}\}$ in the remainder of this paper.

As shown in Figure 2, the ALIGN integrates two cooperative modules, **Analyzer** and **Optimizer**, to generate aligned interfaces. The framework operates through iterative optimization, with each iteration comprising three stages: in Stage 1, the Analyzer identifies agent-environment misalignments by analyzing past interaction trajectories; in Stage 2, the Optimizer generates, validates and refines a new interface based on the detected misalignments; and in Stage 3, the agent interacts with the environment wrapped with the newly generated interface, and the failed task trajectories are fed back to Analyzer for analysis in the next iteration.

3.3 ALIGN Framework

To automate the generation of interfaces that bridge the agent-environment misalignments, ALIGN need to solve two key challenges: (1) how to analyze and identify existing agent-environment misalignments, and (2) how to generate an interface that addresses these misalignments. The overall algorithm process of ALIGN is outlined in Algorithm 1 in Appendix B.

Misalignment Analysis. We represent each agent-environment misalignment using structured text, as shown in the bottom left of Figure 2. The "Agent High-Level Reasoning Intent" and "Environment Rule" respectively depict the agent's expectations of the action and the environment's observation rules, together representing a misalignment. The "Sufficient Observation" represents the observation the environment should provide to resolve the misalignment. To analyze and identify these misalignments, we designed the Analyzer module based on LLMs. In each iteration, the Analyzer takes the failed interaction trajectory $\tau^{(i-1)}$ in the previous iteration, the set of currently identified misalignments $\mathcal{M}$, and the interface $\Phi^{(i-1)}$ from the previous round as input, and generates a new set of misalignments $\mathcal{M}^{(i)}$. The Analyzer needs no human labels or gold trajectories; its supervision is the agent's failed trajectories and the environment's outcome signal (Appendix F.4). Detailed prompts for this process are provided in Appendix F.4.

Interface Generation. Once the new set of misalignments $\mathcal{M}^{(i)}$ is identified, we employ the Optimizer module to generate a new interface. We represent the two modules of the interface, INFERRULES and WRAPSTEP, as Python functions, as shown in the bottom right of Figure 2, to leverage the powerful code generation capabilities of LLMs. In each iteration, the Optimizer takes the newly identified misalignments $\mathcal{M}^{(i)}$ and the previous interface $\Phi^{(i-1)}$ as input, and generates a new interface $\Phi^{(i)}$. The detailed prompts for this process are provided in Appendix F.4.

Experimental Verification. Given the hallucination (Bang et al., 2023; Xu et al., 2024) issues of LLMs, we incorporate an experimental verification procedure. Specifically, after the Analyzer generates $\mathcal{M}^{(i)}$, it will interact with the environment wrapped by the previous interface $\Phi^{(i-1)}$ to validate whether the identified misalignments do indeed exist and can be resolved by the proposed "Sufficient Observation". And after the Optimizer generates the new interface $\Phi^{(i)}$, it will interact with the environment wrapped by this new interface to ensure that the generated interface can resolve the identified misalignments. If the Optimizer finds that the proposed interface is insufficient to address the discovered misalignments, it will provide a refinement strategy and regenerate the interface. This iterative process continues until the interface passes the validation, ensuring that the misalignments identified are appropriately addressed. An example of this process is provided in the bottom center of Figure 2. To facilitate this interaction with the interface-wrapped environment, we designed a set of encapsulated tools for both the Analyzer and Optimizer to use, as described in Appendix F.3.

After each iteration, the agent interacts with the environment wrapped by the new generated interface $\Phi^{(i)}$, and trajectories of the failed tasks are returned to Analyzer for further analysis. The algorithm continues iteratively until one of the following holds: (1) the pre-defined maximum number of iterations is reached; (2) no failed trajectories are produced; (3) no new misalignments are identified.

4 Experiment

4.1 Experimental Settings

Evaluation Protocol. We evaluate each agent in the original environment, then re-evaluate it in the

Method	Interface	Embodied		Web	Tool-use
		ALFWorld	ScienceWorld	WebShop	M^3ToolEval
Vanilla	w/o ALIGN	13.43	14.94	54.10	11.11
	w/ ALIGN	60.45 (+47.02)	27.69 (+12.75)	61.23 (+7.13)	20.83 (+9.72)
ReAct	w/o ALIGN	19.40	20.03	37.20	9.72
	w/ ALIGN	63.43 (+44.03)	28.97 (+8.94)	42.93 (+5.73)	18.06 (+8.34)
Self-Consistency	w/o ALIGN	11.94	14.07	56.23	11.11
	w/ ALIGN	69.40 (+57.46)	25.41 (+11.34)	61.10 (+4.87)	16.67 (+5.56)
Self-Refine	w/o ALIGN	3.73	14.87	44.80	5.55
	w/ ALIGN	40.30 (+36.57)	22.99 (+8.12)	52.30 (+7.50)	6.94 (+1.39)
Planning	w/o ALIGN	9.70	17.13	46.95	11.11
	w/ ALIGN	52.99 (+43.29)	26.34 (+9.21)	54.67 (+7.72)	18.06 (+6.95)

Table 1: **Effect of ALIGN-generated interfaces on four benchmarks.** For every agent we report its score without the interface (w/o ALIGN) and with the interface (w/ ALIGN); the value in parentheses is the absolute improvement. Metrics are task-success rate (%) for ALFWorld and M^3ToolEval, and scores for ScienceWorld and WebShop.

same environment wrapped with ALIGN-generated interface. Interfaces are generated only from training tasks and held fixed during testing.

Benchmarks. We use ALFWorld (Shridhar et al., 2021) for embodied household tasks, ScienceWorld (Wang et al., 2022) for interactive scientific reasoning, WebShop (Yao et al., 2022) for web navigation, and M^3ToolEval (Wang et al., 2024b) for multi-turn tool use.

Agent Methods. We evaluate Vanilla Agent (no specialized prompting), ReAct (Yao et al., 2023) (interleaved reasoning and acting), Self-Consistency (Wang et al., 2023) (majority voting over sampled trajectories), Self-Refine (Madaan et al., 2023) (iterative self-critique) and Planning agent inspired by RAP (Hao et al., 2023) that decomposes tasks into sub-tasks.

Implementation Details. Unless otherwise noted, agents use Qwen2.5-7B-Instruct (Team, 2024) as the base model. Within ALIGN, the Optimizer uses Gemini 2.5 Pro (Google, 2025) for interface code generation, and GPT-4.1 (OpenAI, 2025) is used elsewhere in the Analyzer and Optimizer; the GPT-4.1-series experiments in Section 4.4 instead use the cheaper GPT-4.1-mini for these roles (Appendix C.4). Full setup is in Appendix F.

4.2 Main Results

Table 1 summarizes the task success rates or scores of five representative agent methods in the environment without (w/o) or with (w/) ALIGN-generated interface. The token consumption analysis can be found in Appendix E. Our empirical investigation yields three principal findings:

(1) ALIGN consistently enhances performance across different domains. All evaluated agent methods demonstrate significant performance improvements when utilizing ALIGN-generated interfaces. Specifically, the five agent methods exhibit mean improvements of 45.67% in task-success rate for ALFWorld, 10.07 points for ScienceWorld, 6.59 points for WebShop, and 6.39% in task-success rate for M^3ToolEval.

(2) Agent-environment misalignment is a pervasive phenomenon impeding the agent performance. The observed performance enhancements provide empirical evidence that numerous errors in baseline configurations originate from implicit constraints or under-specified observation, rather than from intrinsic reasoning deficiencies. This finding suggests that when these environmental constraints are explicitly surfaced, agents can execute their intended tasks with substantially improved reliability. Consequently, we posit that agent-environment misalignment is pervasive in interactive decision-making tasks, and addressing this problem is crucial for advancing agent performance.

(3) Alignment between agent and environment can facilitate identification of additional performance-influencing factors. While the Self-Consistency agent achieves a 69.40% success rate in ALFWorld with ALIGN, the performance of Self-Refine agent remains comparatively suboptimal (40.30%), indicating potential deficiencies in the critic and self-refinement capabilities of the Qwen2.5-7B-Instruct model. These limitations are similarly manifested in the M^3ToolEval results. Furthermore, the relatively modest performance improvements in ScienceWorld suggest that Qwen2.5-7B-Instruct may exhibit insufficient sci-

Method	ALFWorld			ScienceWorld		
	w/o ALIGN	w/ ALIGN	Δ	w/o ALIGN	w/ ALIGN	Δ
Vanilla	77.91	26.59	66%	49.12	24.47	50%
ReAct	82.23	38.63	53%	46.61	29.99	36%
Self-Consistency	77.71	15.08	81%	51.10	31.51	38%
Self-Refine	90.38	45.84	49%	58.02	29.48	49%
Planning	74.09	19.14	74%	68.67	20.94	70%
Average	80.46	28.51	65%	54.70	27.28	49%

Table 2: **Impact of the ALIGN-generated interface on consecutive invalid actions.** The metric reports the fraction (%) of consecutive invalid actions. Lower values indicate more desirable behavior. Δ denotes the relative reduction with respect to the **w/o ALIGN** setting.

entific causal reasoning capabilities. These observations indicate that properly aligning agent and environment enables more precise isolation and analysis of other factors influencing agent performance beyond alignment considerations.

4.3 Interface Quality Analysis

Influence on Agent Decision To quantitatively assess the influence of ALIGN-generated interfaces on agent decision beyond end-task performance metric, we introduce a metric that measures the frequency of *consecutive invalid actions* by calculating the proportion of the actions that occur within sequences of two or more consecutive `invalid` steps. Lower values of this metric indicate: (1) enhanced agent awareness of implicit preconditions, and (2) improved recovery capability following isolated errors. Table 2 presents the results for five agent methods implemented on ALFWorld and ScienceWorld. The empirical results demonstrate a substantial reduction in consecutive invalid actions frequency across all agent methods when utilizing ALIGN-generated interfaces. Specifically, we observe a mean reduction of 65% in ALFWorld and 49% in ScienceWorld. These findings provide evidence that ALIGN effectively enriches the information conveyed by the observation, preventing agents from entering repetitive error cycles, which aligns with the findings documented in Section 4.2.

Comparison with Agentic Systems and Human-designed Interfaces To further assess the effectiveness of our automatically generated interfaces, we compare ALIGN against (1) agentic frameworks equipped with carefully designed reasoning, planning and memory modules and (2) human-designed interfaces. The experimental setup and results are presented in Appendix C.2. As shown in Table 8, even without bespoke reasoning, planning, or memory modules, a vanilla agent that directly outputs the next action yields a 6.71 percentage points higher success rate than the best agentic system when paired with ALIGN-generated interfaces, indicating agent-environment misalignment substantially constrains the performance of LLM-based agents in interactive tasks. Moreover, using interfaces automatically generated by ALIGN yields a 13.44 percentage points higher than human-designed interfaces, further validating the effectiveness of our method (Table 9).

4.4 Generalization and Generality Study

Generalization Study To evaluate the generalization capabilities of ALIGN, we performed the following two experiments, with the results presented in Table 3 and detailed in Appendix C.3.

(a) Interface source: Vanilla agent				
Target method	ALF.	Sci.	Web.	M^3T.
ReAct	+39.56	+12.29	+7.87	+5.56
Self-Consistency	+51.49	+15.30	+3.00	+8.33
Self-Refine	+34.33	+14.11	+6.17	+4.17
Planning	+41.05	+9.66	+3.26	+11.11
(b) Interface source: Qwen2.5-7B-Instruct agent				
Target LLM	ALF.	Sci.	Web.	M^3T.
Qwen2.5-14B-Instruct	+17.46	+4.61	+4.66	+6.11
Llama3.1-8B-Instruct	+5.97	+10.27	+0.33	+0.83
Llama3.3-70B-Instruct	+5.82	+3.99	+5.68	+1.67

Table 3: **Generalization of ALIGN-generated interfaces across agents and models.** Mean performance improvements from applying ALIGN-generated interfaces in the four environments across different settings. (a) Cross-agent transfer: interfaces generated with a Vanilla agent improve other agent methods. (b) Cross-model transfer: interfaces generated with Qwen2.5-7B-Instruct can generalize to other LLMs.

(1) ALIGN can generalize to different agent architectures. Panel (a) of Table 3 applies interfaces generated with the Vanilla agent to the other four agents. Across all four environments every target agent shows consistent growth, demonstrating that ALIGN captures genuine and previously unex-

posed environment constraints. This also reinforces the earlier conclusion that agent-environment misalignment is a pervasive source of error independent of the agent's reasoning style.

(2) ALIGN can generalize to larger and heterogeneous LLMs. Panel (b) of Table 3 examines whether an interface generated with Qwen2.5-7B-Instruct can extend to larger or architecturally different model backbones. The results demonstrate that ALIGN-generated interfaces lead to performance improvements across base models of varying sizes and architectural families, which indicates that our method possesses strong generalization capabilities. We also observe that this generalization is not uniformly robust across all model families and datasets. For instance, Llama3.1-8B-Instruct (Meta, 2025a) shows only a marginal gain of +0.33 on the WebShop benchmark. This limited improvement may be attributed to the inherent reasoning capabilities of the model itself.

Taken together, these results show that ALIGN-generated interfaces can generalize (1) across agent policies and (2) across model scales and families, validating the practicality of ALIGN.

(a) GPT-4.1 series		
Base Model	Interface	SR (%)
GPT-4.1-mini	w/o ALIGN	28.36
	w/ ALIGN	64.93 (+36.57)
GPT-4.1	w/o ALIGN	73.88
	w/ ALIGN	93.28 (+19.40)
(b) GiGPO-Qwen2.5-7B-Instruct		
Agent Method	Interface	SR (%)
Vanilla	w/o ALIGN	35.04
	w/ ALIGN	55.97 (+20.93)
Training Config	w/o ALIGN	89.55
	w/ ALIGN	92.54 (+2.99)

Table 4: **Generality of ALIGN.** Task success rates (SR) without and with ALIGN-generated interfaces in ALFWorld across two settings. (a) Using GPT-4.1 series models as the base model; (b) Using GiGPO-Qwen2.5-7B-Instruct under different agent architectures.

Generality Study In this work, our empirical observations indicate that the root cause of agent-environment misalignment lies in the robustness of the interface itself, making it a universal issue that affects agents irrespective of the underlying model capability. To further validate this claim and assess the generality of ALIGN, we conduct experiments on both closed-source LLMs and domain-specific models trained within the environment. For the former, we use the GPT-4.1 series; for the latter, we use GiGPO-Qwen2.5-7B-Instruct-ALFWorld (Feng et al., 2025), a state-of-the-art model specifically post-trained on ALFWorld via reinforcement learning. Detailed experimental setup and full results are provided in Appendix C.4. As the results reported in Panel (a) of Table 4 shown, applying the ALIGN-generated interface substantially improves the performance of the GPT-4.1-based agent from 73.88% to 93.28%. Meanwhile, as the results reported in Panel (b) of Table 4 shown, the ALIGN-generated interface also enhances the performance of the domain-specific model under both our Vanilla Agent setting and its original training configuration, from 35.04% to 55.97% and 89.55% to 92.54%, respectively. These findings demonstrate that the fundamental and pervasive nature of agent-environment misalignment stems from deficiencies in the environment's interface rather than solely from the reasoning limitations of any given model, and further corroborate the generality of our method across both frontier and domain-specialized models.

4.5 Ablation Study

Ablation on Interface Components Starting from the full ALIGN interface, we ablate InferRules and WrapStep separately. As shown in Table 5, both ablations reduce performance: w/o InferRules averages -6.72 percentage points on ALFWorld and -2.05 on ScienceWorld, while removing WrapStep yields a larger decline of -31.79 percentage points and -7.84, respectively. These decreases confirm that each interface component contributes meaningfully. Moreover, the much larger drop w/o WrapStep shows the critical role of fine-grained and enriched observation during interaction. This also suggests that rich, LLM-friendly observation should be prioritized by future environment designers when constructing environments.

Ablation on Experimental Verification To assess whether the experimental verification procedure in Section 3.3 is indispensable, we ablated it with the Vanilla agent on ALFWorld. As a surrogate, the Analyzer and Optimizer each sampled six candidates per iteration and selected their top one, under decoding temperatures $T \in \{0.2, 0.5\}$. As shown in Table 6, success rate deteriorates sharply across iterations, reflecting the limited single-shot reliability of LLMs in diagnosing misalignments and synthesizing interfaces, and confirming the necessity of experimental verification.

Method	w/o InferRules		w/o WrapStep	
	ALF.	Sci.	ALF.	Sci.
Vanilla	-8.96	-3.35	-33.58	-4.72
ReAct	-5.22	-2.08	-17.91	-6.44
Self-Consistency	-1.49	-2.30	-46.27	-10.59
Self-Refine	-7.46	-1.72	-34.33	-7.59
Planning	-10.45	-0.78	-26.87	-9.86
Mean	-6.72	-2.05	-31.79	-7.84

Table 5: **Ablation on Interface components.** Values show performance change on ALFWorld and ScienceWorld. Negative values mean drops from *Full* interface.

Temp.	Iter0	Iter1	Iter2	Iter3
0.2	13.43	22.39	0.00	0.00
0.5	13.43	23.88	1.49	0.75

Table 6: Task success rate (%) on ALFWorld across iterations without experimental verification procedure.

4.6 Further Analyses

Complementarity with Agent-Side Rule Learning We test the complementarity claimed in Section 2 by injecting the ten rules released by AutoManual (Chen et al., 2024a) into the agent prompt, exactly as its testing stage consumes a manual (adaptation details in Appendix D.2). Table 7 reports results for Qwen2.5-7B-Instruct and Qwen3-8B (Yang et al., 2025), the latter reusing the Qwen2.5-7B-Instruct interface (cf. Table 3(b)). The combination is the best configuration for both models, and the two methods repair different levels of failure: AutoManual's rules operate at the plan level (task-type recipes), whereas ALIGN operates at the execution level (why a step failed and which precondition applies).

Content Audit of Injected Observations Interfaces are generated from training tasks only and frozen at test time, so the remaining question is one of content. We audited all 312 agent-facing strings injected by the 20 final interfaces (Appendix D.3): an injection is *mechanic-level* iff it is a deterministic function of the current state and the attempted action and contains no task-goal content. ALFWorld and WebShop contain no task-specific guidance, so the headline results rest on mechanic-level content only; a small minority in ScienceWorld and M^3ToolEval (21 of 312 strings) encodes procedure-level knowledge, which we trace to their one- or two-task training splits and mitigate with a task-agnosticism constraint in the Optimizer prompt (Appendix F.4).

Exploration and Strategy Diversity Analyzing the complete evaluation trajectories (Appendix D.4), we find that action-type entropy stays in the same band with and without ALIGN, that agents visit more distinct locations per task with ALIGN, and that the gains consist of failure-to-success conversions (50–79 tasks per method on ALFWorld) against only 1–7 regressions: the signature of restored observability rather than lowered difficulty.

Base Model	Baseline	+AutoManual	+ALIGN	+Both
Qwen2.5-7B-Instruct	13.43	29.10	60.45	**64.93**
Qwen3-8B	15.67	52.24	47.76	**53.73**

Table 7: **Complementarity of ALIGN with agent-side rule learning.** Task success rates (%) of the Vanilla agent on ALFWorld with AutoManual's released rules, the ALIGN-generated interface, and both; Qwen3-8B reuses the Qwen2.5-7B-Instruct interface.

Generator Capability and Cost Generation is a one-time cost per environment: the full $K=8$ iterations on ALFWorld cost about USD 9.4 at list prices, i.e., about USD 0.003 per task-run over the reuse in this paper, while the interface itself involves no model calls and reduces per-task inference to 58% of the baseline's input tokens and 43% of its LLM calls (Appendix E.1). Only the Optimizer's code synthesis requires a frontier model: with GPT-4.1-mini as the Analyzer and verification model, the generated interface is no worse than that of the frontier stack (64.93% vs. 60.45% for Qwen2.5-7B-Instruct), whereas a fully self-sufficient Qwen2.5-7B-Instruct stack fails to complete generation (Appendix D.5); the agent itself has no such floor.

5 Conclusion

In this work, we introduce **ALIGN**, a framework that automatically generates aligned interfaces to alleviate the **agent-environment misalignment** in interactive decision-making tasks. By diagnosing implicit constraints via the Analyzer and synthesizing interfaces via the Optimizer, ALIGN achieves significant improvements across four benchmarks spanning embodied tasks, web navigation, and tool-use. Our results show that ALIGN generalizes to unseen models and strategies, offering a robust, plug-and-play solution that decouples agent design from manual environment-specific alignment.

Limitations

Despite the effectiveness of ALIGN in alleviating agent-environment misalignment, this work repre-

sents an initial exploration into automated interface generation, and several boundaries should be stated plainly.

Scope: text-based environments with a programmable boundary. All of our evaluation is conducted in text-based environments with discrete action spaces, spanning embodied tasks, web navigation, and tool use. ALIGN applies wherever the agent–system boundary is programmable, which holds for tool-use, API, web, and code agents, whose production systems are already mediated by hand-built wrapper layers such as tool schemas and agent-computer interfaces (Yang et al., 2024b); physical robots and raw pixel streams fall outside this condition. Although the formalization in Section 3.1 is modality-agnostic, and the textual projections perceived by current GUI agents (accessibility trees and Set-of-Marks element lists (Yang et al., 2023a)) would allow INFERRULES and WRAPSTEP to apply as they are, we claim no GUI results: pixel-level observation enrichment remains an open problem, and the accessibility-tree instantiation is the natural next step.

Supervision and undiagnosable causes. The Analyzer requires the environment's success/failure signal and the agent's own failed trajectories, but no human labels or gold trajectories; settings with no outcome signal at all are not covered. Moreover, every hypothesized misalignment must be experimentally confirmed before an interface change is accepted, so a cause that the Analyzer cannot verify (e.g., sensor noise or undocumented API behavior) is discarded and the interface simply remains unchanged: the failure mode is graceful, but no improvement is obtained.

Over-guidance and cross-family robustness. The content audit in Appendix D.3 finds a small amount of procedure-level content in the two domains with one- or two-task training splits; we mitigate this with a task-agnosticism constraint in the Optimizer, but larger training splits would be a more principled remedy. In addition, an interface generated with one model family may not recognize the action surface forms of another (e.g., Qwen3-8B emits "use desklamp 1 on X" where Qwen2.5 emits "use desklamp 1"), so generating family-general interfaces is a direction for future work.

Metrics for interface quality. Beyond task success rates and consecutive invalid actions, we report behavioral diagnostics on action diversity and exploration in Appendix D.4. More comprehensive metrics, such as LLM-as-a-Judge (Zheng et al., 2023) evaluation of whether an interface conveys environment constraints, remain a promising direction.

Acknowledgments

This work is supported by Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM101), the National Natural Science Foundation of China (No. 62276152, 62236011), funding from Wuxi Research Institute of Applied Technologies, Tsinghua University under Grant 20242001120, and Key Laboratory of Ethnic Language Intelligent Analysis and Security Governance of MOE, Minzu University of China, Beijing, China.

References

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2024. Agent S: an open agentic framework that uses computers like a human. *CoRR*, abs/2410.08164.

Hao Bai, Yifei Zhou, Erran Li Li, Sergey Levine, and Aviral Kumar. 2025. Digi-Q: Transforming VLMs to device-control agents via value-based offline RL.

Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. 2023. A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, IJCNLP 2023 -Volume 1: Long Papers, Nusa Dua, Bali, November 1 - 4, 2023*, pages 675–718. Association for Computational Linguistics.

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence Illing Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphaël Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, and 5 others. 2024. Jumanji: a diverse suite of scalable reinforcement learning environments in JAX. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI gym. *CoRR*, abs/1606.01540.

Timothy Bula, Saurabh Pujar, Luca Buratti, Mihaela Bornea, and Avirup Sil. 2025. SeaView: Software engineering agent visual interface for enhanced workflow. *arXiv preprint arXiv:2504.08696*.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. FireAct: Toward language agent fine-tuning. *CoRR*, abs/2310.05915.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. 2024a. AutoManual: Constructing instruction manuals by LLM agents via interactive environmental learning. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024b. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 9354–9366. Association for Computational Linguistics.

Zhixun Chen, Ming Li, Yuxuan Huang, Yali Du, Meng Fang, and Tianyi Zhou. 2025. ATLaS: Agent tuning via learning critical steps. *CoRR*, abs/2503.02197.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for LLM agent training. *CoRR*, abs/2505.10978.

Peiyuan Feng, Yichen He, Guanhua Huang, Yuan Lin, Hanchong Zhang, Yuchen Zhang, and Hang Li. 2024. AGILE: A novel reinforcement learning framework of LLM agents. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

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. *CoRR*, abs/2501.01702.

Yingqiang Ge, Wenyue Hua, Kai Mei, Jianchao Ji, Juntao Tan, Shuyuan Xu, Zelong Li, and Yongfeng Zhang. 2023. OpenAGI: When LLM meets domain experts. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Google. 2025. Gemini 2.5 Pro preview model card.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2024. Navigating the digital world as humans do: Universal visual grounding for GUI agents. *CoRR*, abs/2410.05243.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 8048–8057. ijcai.org.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 8154–8173. Association for Computational Linguistics.

Kaiyu He, Mian Zhang, Shuo Yan, Peilin Wu, and Zhiyu Zoey Chen. 2024. IDEA: Enhancing the rule learning ability of large language model agent through induction, deduction, and abduction. *Preprint*, arXiv:2408.10455.

Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. 2017. AI2-THOR: an interactive 3d environment for visual AI. *CoRR*, abs/1712.05474.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Xuanyu Lei, Zonghan Yang, Xinrui Chen, Peng Li, and Yang Liu. 2025. Scaffolding coordinates to promote vision-language coordination in large multi-modal models. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 2886–2903. Association for Computational Linguistics.

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. 2024. AgentBench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024. OmniParser for pure vision based GUI agent. *CoRR*, abs/2408.00203.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information*

Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Meta. 2025a. Model cards and prompt formats Llama 3.1.

Meta. 2025b. Model cards and prompt formats Llama 3.3.

Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. 2024. ScreenAgent: A vision language model-driven computer control agent. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 6433–6441. ijcai.org.

OpenAI. 2025. Introducing GPT-4.1 in the api.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023, San Francisco, CA, USA, 29 October 2023- 1 November 2023*, pages 2:1–2:22. ACM.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, Singapore. Association for Computational Linguistics.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, Tianjie Zhang, Wei Xu, Jie Tang, and Yuxiao Dong. 2024. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. *CoRR*, abs/2411.02337.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, and 16 others. 2025. UI-TARS: pioneering automated GUI interaction with native agents. *CoRR*, abs/2501.12326.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E. Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy P. Lillicrap, and Oriana Riva. 2024. AndroidWorld: A dynamic benchmarking environment for autonomous agents. *CoRR*, abs/2405.14573.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. 2025. Agentsquare: Automatic LLM agent search in modular design space. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. 2021. ALFWorld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. 2024. OS-Genesis: Automating GUI agent trajectory construction via reverse task synthesis. *CoRR*, abs/2412.19723.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Mark Towers, Ariel Kwiatkowski, Jordan K. Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. 2024. Gymnasium: A standard interface for reinforcement learning environments. *CoRR*, abs/2407.17032.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024.

Ruoyao Wang, Peter A. Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 11279–11298. Association for Computational Linguistics.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better LLM agents. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. 2024c. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Monica Lam, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. 2025. RAGEN: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *Preprint*, arXiv:2504.20073.

Licheng Wen, Daocheng Fu, Xin Li, Xinyu Cai, Tao Ma, Pinlong Cai, Min Dou, Botian Shi, Liang He, and Yu Qiao. 2024. Dilu: A knowledge-driven approach to autonomous driving with large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Ziwei Xu, Sanjay Jain, and Mohan S. Kankanhalli. 2024. Hallucination is inevitable: An innate limitation of large language models. *CoRR*, abs/2401.11817.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024a. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023a. Set-of-Mark prompting unleashes extraordinary visual grounding in GPT-4V. *CoRR*, abs/2310.11441.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024b. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Zeyuan Yang, Peng Li, and Yang Liu. 2023b. Failures pave the way: Enhancing large language models through tuning-free rule accumulation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 1751–1777. Association for Computational Linguistics.

Zonghan Yang, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. 2024c. ReAct meets ActRe: When language agents enjoy training data autonomy. *CoRR*, abs/2403.14589.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Junchi Yu, Ran He, and Zhitao Ying. 2024. Thought propagation: an analogical approach to complex reasoning with large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual*

meeting, August 11-16, 2024, pages 3053–3077. Association for Computational Linguistics.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. GPT-4V(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

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. 2024a. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Siyu Zhou, Tianyi Zhou, Yijun Yang, Guodong Long, Deheng Ye, Jing Jiang, and Chengqi Zhang. 2024b. WALL-E: world alignment by rule learning improves world model-based LLM agents. *CoRR*, abs/2410.07484.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. 2024c. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*.

A LLM Usage Statement

Throughout the completion of this work, the LLM was employed solely for the purpose of refining sentences and improving grammatical accuracy during the manuscript writing process.

B Formalization of the ALIGN Algorithm

The formalization of the ALIGN algorithm is outlined in Algorithm 1.

Algorithm 1 ALIGN: Auto-Aligned Interface Generation

Require: Environment $\mathcal{E}$, Agent π, Task training set $\mathcal{T}_{\text{train}}$, Maximum iterations K

Initialize misalignment set $\mathcal{M} \leftarrow \emptyset$, interface $\Phi^{(0)} \leftarrow \{\textsc{InferRules}^{(0)}, \textsc{WrapStep}^{(0)}\}$, where $\textsc{InferRules}^{(0)}$ and $\textsc{WrapStep}^{(0)}$ are identity functions
for $i = 1, 2, \ldots, K$ **do**
  $\tilde{\mathcal{E}}^{(i-1)} \leftarrow$ Environment $\mathcal{E}$ wrapped with interface $\Phi^{(i-1)}$
  $\tau_{\text{fail}}^{(i-1)} \leftarrow$ Failed trajectories from agent π interacting with $\tilde{\mathcal{E}}^{(i-1)}$ on $\mathcal{T}_{\text{train}}$
  if $\tau_{\text{fail}}^{(i-1)} = \emptyset$ **then**
    break ▷ No more failures in the training set
  end if
  // Stage 1: Misalignment Analysis
  $\mathcal{M}^{(i)} \leftarrow \text{Analyzer}(\tau_{\text{fail}}^{(i-1)}, \mathcal{M}, \Phi^{(i-1)})$
  if $\mathcal{M}^{(i)} = \emptyset$ **then**
    break ▷ No new misalignments identified
  end if
  $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{M}^{(i)}$
  // Stage 2: Interface Generation
  $\Phi^{(i)} \leftarrow \text{Optimizer}(\mathcal{M}^{(i)}, \Phi^{(i-1)})$
end for
return final interface $\Phi^{(i)}$

C Supplementary Experimental Setup and Detailed Results

C.1 Preliminary Experiments

To preliminarily assess the significance of agent-environment misalignment, we conducted exploratory experiments on the ALFWorld. We employed the vanilla Qwen2.5-7B-Instruct agent with a temperature setting of 0.0. The deployment protocol, prompt template, followed the same configuration described in Appendix F and Appendix F.4.

During the experiments, we introduced a minor modification to the environment: if the agent issued the action *examine receptacle* and the environment returned the default observation "Nothing happens.", we replaced it with "You need to first go to receptacle before you can examine it." This simple adjustment increased the agent's task success rate from 13.4% to 31.3%.

C.2 Interface Quality Analysis Experiments

To further assess the quality of the ALIGN-generated interface, we first compare our method with human-designed agentic system. Our experiments are conducted on ALFWorld using the AgentSquare (Shang et al., 2025) framework. To maximize the advantages of the agentic system, we adopt gpt-4.1-2025-04-14 as the base model, select OPENAGI (Ge et al., 2023) for the planning module, Self-Refine (Madaan et al., 2023) for the reasoning module, and evaluate memory using Generative (Park et al., 2023), DiLu (Wen et al., 2024), TP (Yu et al., 2024), and VOYAGER (Wang et al., 2024a). For our approach, we employ a gpt-4.1-2025-04-14-based vanilla agent, where the interface is generated with the gpt-4.1-mini-2025-04-14-based vanilla agent by ALIGN (the experimental setup is same as Appendix C.4). The results are reported in Table 8.

Furthermore, we compare the ALIGN-generated interface against the human-designed interface. We adopt the following configurations for comparison with our method: (1) **Few-shot:** Settings identical to those in the ReAct (Yao et al., 2023); (2) **Valid Actions:** Supplying the agent with all valid actions at every response turn, analogous to the `check_valid_actions` configuration in AgentBoard (Ma et al., 2024); (3) **Human-Designed Interface:** Interfaces manually crafted by Ph.D. students after inspecting ALFWorld experiments, examining trajectories, and running experiments themselves. The design logic includes: executing "go to" prior to each action; automatically checking object labels; converting "put" to "move" when appropriate; returning the action space upon invalid actions; issuing reminders when "clean with" is applied to non-sinkbasin objects; and other hand-engineered rules. We use Qwen2.5-7B-Instruct as the base model. Experimental results are reported in Table 9.

C.3 Generalization Study Experiments

Detailed results of the generalization study are provided for the cross-method experiments in Table 10 and for the cross-model experiments in Tables 11, 12, and 13.

C.4 Generality Study Experiments

For the validation on closed-source LLMs, we selected the GPT-4.1 family, experimenting with gpt-4.1-mini-2025-04-14 and gpt-4.1-2025-04-14. Different from the main experiments, this setting also uses a cheaper generation stack: gpt-4.1-mini-2025-04-14 replaces GPT-4.1 both as the Analyzer and as the experimental-verification model of the Optimizer, while Gemini 2.5 Pro is still used for interface code synthesis. We used a gpt-4.1-mini-based Vanilla agent to synthesize the interface with ALIGN, and then applied the same interface to agents powered by gpt-4.1-mini and gpt-4.1. All other experimental settings were identical to those in the main experiments. The results are presented in Table 14; the same generation stack is analyzed further in Appendix D.5.

For domain-specific models trained within the environment, we used GiGPO-Qwen2.5-7B-Instruct-ALFWorld, a state-of-the-art model post-trained on ALFWorld via reinforcement learning (Feng et al., 2025). We reused the interface produced in our main experiment (generated with the base Qwen2.5-7B-Instruct model under the Vanilla Agent method). At evaluation time, we considered two configurations: (1) our Vanilla Agent setting, and (2) a configuration that matches the logic and prompt setting used during training in the original paper.

C.5 Ablation Study Experiments

The full result of interface ablation experiment can be found in Table 15.

D Additional Analyses

D.1 Positioning against Rule-Learning and Prompt-Optimization Methods

Table 16 summarizes the boundary between ALIGN and the two families of methods that are structurally closest to it. WALL-E (Zhou et al., 2024b) aligns the agent's internal world model by learning rules from collected trajectories, and AutoManual (Chen et al., 2024a) builds a manual that conditions its planner; APO methods (Zhou et al., 2023; Pryzant et al., 2023; Yang et al., 2024a; Wang et al., 2024c) search over the agent's prompt with an LLM-driven optimizer and select candidates by held-out scoring. All of them leave the environment's feedback channel, the source of the state aliasing studied in this paper, untouched, whereas ALIGN repairs the channel itself. Two consequences follow. First, the artifact is reusable below the agent abstraction: one interface serves five agent architectures (Table 3(a)), four model backbones (Table 3(b)), and an RL-specialized

Agent Framework	Interface	Memory Module	pick and place	pick clean and place	pick heat and place	pick cool and place	look at / examine in light	pick two obj and place	Success Rate (%)
AgentSquare	/	Generative	95.83	87.10	69.57	95.24	83.33	88.24	86.57
AgentSquare	/	DiLu	91.67	87.10	52.17	95.24	83.33	70.59	80.60
AgentSquare	/	TP	87.50	51.61	4.35	61.90	27.78	47.06	47.76
AgentSquare	/	VOYAGER	95.83	83.87	52.17	90.48	83.33	64.71	79.10
Vanilla Agent	w/o ALIGN	/	100.00	93.55	13.04	71.43	61.11	100.00	73.88
Vanilla Agent	w/ ALIGN	/	100.00	100.00	78.26	100.00	77.78	100.00	**93.28**

Table 8: Experimental results of the comparison between agents with ALIGN-generated interface and agents with human-designed reasoning, planning and memory module.

Experimental Setting	Success Rate (%)
w/o Interface	13.43
Few-shot	44.78
Valid Actions	44.03
Human Designed Interface	47.01
ALIGN-generated Interface	**60.45**

Table 9: Experimental results of the comparison between agents with ALIGN-generated interface and agents with human-designed interfaces.

policy at evaluation time (Table 4(b)). AutoManual's manual also transfers between models, but as prompt content for a compatible planner, whereas the ALIGN wrapper needs no agent-side compatibility at all. Second, the Analyzer–Optimizer loop is not APO with different variables: its search space is environment-side code, and its acceptance signal is executed experiments rather than a held-out score (Appendix F.3). Table 6 shows that this signal is load-bearing, as success collapses across iterations without it.

On the absolute numbers reported by these works (95% on ALFWorld for WALL-E and 97.4% for AutoManual): they rest on GPT-3.5/GPT-4-class base models together with agent-side scaffolds, whereas our main table uses Qwen2.5-7B-Instruct. Under a comparable base model, a vanilla agent with the ALIGN-generated interface reaches 93.28% (Table 8), above the best agentic system we tested (AgentSquare, 86.57%). Because the two directions are complementary rather than competing, we directly test their combination in Appendix D.2.

D.2 Complementarity with Agent-Side Rule Learning

Setup. We use the ten rules that AutoManual's GPT-4-based Builder evolved in its original ALFWorld experiments, as shipped in its public repository, and inject them into the agent prompt exactly as AutoManual's testing stage consumes a manual. Owing to differences in ALFWorld versions and implementations, four mechanical adaptations were applied with the semantics left untouched: (1) the Python helper `find_object` was replaced by an equivalent textual description of the search procedure; (2) "put X in/on Y" was rewritten as "move X to Y" to match our action space; (3) references to AutoManual's build log (e.g., "as observed in epoch_14") were removed; and (4) the naming style was changed from `desk_1` to "desk 1". The base models are Qwen2.5-7B-Instruct and Qwen3-8B (Yang et al., 2025), both under the Vanilla agent; Qwen3-8B reuses the interface generated with the Qwen2.5-7B-Instruct agent, i.e., the cross-model transfer setting of Table 3(b). All other settings follow the main experiments.

Results. As shown in Table 7, the combination is the best configuration for both model families (64.93% and 53.73%), and the two methods repair different levels of failure. AutoManual's rules operate at the plan level, encoding task-type recipes, whereas ALIGN operates at the execution level, telling the agent why a step failed and which precondition applies. Qwen2.5-7B-Instruct mostly knows what to do but fails in execution, as its issued actions diverge from its intent, so ALIGN contributes more (+47.02 vs. +15.67 points); Qwen3-8B shows the reverse profile (+32.09 vs. +36.57 points). We also observed cross-family behavioral drift that affects the transferred interface on Qwen3-8B: for instance, Qwen2.5 emits "use desklamp 1", which the interface recognizes, whereas Qwen3 emits "use desklamp 1 on X", which it does not. The transfer nevertheless remains strong (+32.09 points on Qwen3-8B), and generating more robust, family-general interfaces is a direction for future

Base Method: Vanilla		Embodied		Web	Tool-use
Method	**Interface**	**ALFWorld**	**ScienceWorld**	**WebShop**	**M^{3}ToolEval**
ReAct	w/o ALIGN	19.40	20.03	37.20	9.72
	w/ ALIGN	58.96 (+39.56)	32.32 (+12.29)	45.07 (+7.87)	15.28 (+5.56)
Self-Consistency	w/o ALIGN	11.94	14.07	56.23	11.11
	w/ ALIGN	63.43 (+51.49)	29.37 (+15.30)	59.23 (+3.00)	19.44 (+8.33)
Self-Refine	w/o ALIGN	3.73	14.87	44.80	5.55
	w/ ALIGN	38.06 (+34.33)	28.98 (+14.11)	50.97 (+6.17)	9.72 (+4.17)
Planning	w/o ALIGN	9.70	17.13	46.95	11.11
	w/ ALIGN	50.75 (+41.05)	26.79 (+9.66)	50.21 (+3.26)	22.22 (+11.11)

Table 10: **Generalization of ALIGN-generated interfaces generated with Vanilla agents to other agent methods.** For each agent we report its score without the interface (w/o ALIGN) and with the interface (w/ ALIGN); the value in parentheses is the *absolute* improvement.

Base Model: Qwen2.5-14B-Instruct		Embodied		Web	Tool-use
Method	**Interface**	**ALFWorld**	**ScienceWorld**	**WebShop**	**M^{3}ToolEval**
Vanilla	w/o ALIGN	48.51	22.58	53.67	13.89
	w/ ALIGN	52.24 (+3.73)	37.58 (+15.00)	58.40 (+4.73)	18.06 (+4.17)
ReAct	w/o ALIGN	54.48	31.24	39.73	15.28
	w/ ALIGN	70.15 (+15.67)	29.79 (-1.45)	42.17 (+2.44)	26.39 (+11.11)
Self-Consistency	w/o ALIGN	43.28	25.60	52.63	13.89
	w/ ALIGN	72.39 (+29.11)	26.68 (+1.08)	51.07 (-1.56)	27.78 (+13.89)
Self-Refine	w/o ALIGN	5.22	18.97	41.00	15.28
	w/ ALIGN	14.18 (+8.96)	20.72 (+1.75)	39.93 (-1.07)	16.67 (+1.39)
Planning	w/o ALIGN	49.25	21.46	31.72	25.00
	w/ ALIGN	79.10 (+29.85)	28.13 (+6.67)	50.47 (+18.75)	25.00 (0.00)

Table 11: **Generalization of ALIGN-generated interfaces generated with Qwen2.5-7B-Instruct to Qwen2.5-14B-Instruct.** For each agent we report its score without the interface (w/o ALIGN) and with the interface (w/ ALIGN); the value in parentheses is the *absolute* improvement.

Base Model: Llama3.1-8B-Instruct		Embodied		Web	Tool-use
Method	**Interface**	**ALFWorld**	**ScienceWorld**	**WebShop**	**M^{3}ToolEval**
Vanilla	w/o ALIGN	5.22	23.59	35.17	5.56
	w/ ALIGN	14.18 (+8.96)	36.40 (+12.81)	24.00 (-11.17)	1.39 (-4.17)
ReAct	w/o ALIGN	1.49	22.42	27.12	12.50
	w/ ALIGN	15.67 (+14.18)	28.74 (+6.32)	27.10 (-0.02)	22.22 (+9.72)
Self-Consistency	w/o ALIGN	5.22	25.21	29.80	4.17
	w/ ALIGN	11.94 (+6.72)	34.83 (+9.62)	15.83 (-13.97)	2.78 (-1.39)
Self-Refine	w/o ALIGN	0.00	22.34	27.70	1.39
	w/ ALIGN	0.75 (+0.75)	31.33 (+8.99)	37.43 (+9.73)	1.39 (0.00)
Planning	w/o ALIGN	6.72	13.33	23.67	4.17
	w/ ALIGN	5.97 (-0.75)	26.95 (+13.62)	40.77 (+17.10)	4.17 (0.00)

Table 12: **Generalization of ALIGN-generated interfaces generated with Qwen2.5-7B-Instruct to Llama3.1-8B-Instruct.** For each agent we report its score without the interface (w/o ALIGN) and with the interface (w/ ALIGN); the value in parentheses is the *absolute* improvement.

Base Model: Llama3.3-70B-Instruct		Embodied		Web	Tool-use
Method	**Interface**	**ALFWorld**	**ScienceWorld**	**WebShop**	**M^{3}ToolEval**
Vanilla	w/o ALIGN	52.99	55.77	51.67	37.50
	w/ ALIGN	43.28 (-9.71)	57.74 (+1.97)	62.07 (+10.40)	33.33 (-4.17)
ReAct	w/o ALIGN	45.52	56.50	58.22	34.72
	w/ ALIGN	47.01 (+1.49)	58.28 (+1.78)	53.83 (-4.39)	43.06 (+8.34)
Self-Consistency	w/o ALIGN	54.48	56.66	50.37	36.11
	w/ ALIGN	65.67 (+11.19)	59.24 (+2.58)	55.63 (+5.26)	34.72 (-1.39)
Self-Refine	w/o ALIGN	38.06	56.97	38.40	1.39
	w/ ALIGN	46.27 (+8.21)	60.17 (+3.20)	47.85 (+9.45)	0.00 (-1.39)
Planning	w/o ALIGN	58.96	48.75	54.90	33.33
	w/ ALIGN	76.87 (+17.91)	59.17 (+10.42)	62.60 (+7.70)	40.28 (+6.95)

Table 13: **Generalization of ALIGN-generated interfaces generated with Qwen2.5-7B-Instruct to Llama3.3-70B-Instruct.** For each agent we report its score without the interface (w/o ALIGN) and with the interface (w/ ALIGN); the value in parentheses is the *absolute* improvement.

Base Model	**Interface**	**pick and place**	**pick clean and place**	**pick heat and place**	**pick cool and place**	**look at / examine in light**	**pick two obj and place**	**Success Rate (%)**
gpt-4.1-mini	w/o ALIGN	58.33	22.58	8.70	9.52	22.22	52.94	28.36
	w/ ALIGN	95.83	87.10	26.09	80.95	27.78	52.94	**64.93**
gpt-4.1	w/o ALIGN	100.00	93.55	13.04	71.43	61.11	100.00	73.88
	w/ ALIGN	100.00	100.00	78.26	100.00	77.78	100.00	**93.28**

Table 14: Experimental results for GPT-4.1 series agents with ALIGN on ALFWorld.

Method	**Interface**	**Embodied**		**Web**	**Tool**
		ALFWorld	**ScienceWorld**	**Webshop**	**M^{3}ToolEval**
Vanilla	Full	60.45	27.69	61.23	20.83
	w/o INFERRULES	51.49 (-8.96)	24.34 (-3.35)	51.03 (-10.20)	18.06 (-2.77)
	w/o WRAPSTEP	26.87 (-33.58)	22.97 (-4.72)	61.23 (-0.00)	11.11 (-9.72)
ReAct	Full	63.43	28.97	42.93	18.06
	w/o INFERRULES	58.21 (-5.22)	26.89 (-2.08)	35.97 (-6.96)	9.72 (-8.34)
	w/o WRAPSTEP	45.52 (-17.91)	22.53 (-6.44)	47.60 (+4.67)	19.44 (+1.38)
Self-Consistency	Full	69.40	25.41	61.10	16.67
	w/o INFERRULES	67.91 (-1.49)	23.11 (-2.30)	55.67 (-5.43)	13.89 (-2.78)
	w/o WRAPSTEP	23.13 (-46.27)	14.82 (-10.59)	60.67 (-0.43)	15.28 (-1.39)
Self-Refine	Full	40.30	22.99	52.30	6.94
	w/o INFERRULES	32.84 (-7.46)	21.27 (-1.72)	46.33 (-5.97)	6.94 (-0.00)
	w/o WRAPSTEP	5.97 (-34.33)	15.40 (-7.59)	47.80 (-4.50)	6.94 (-0.00)
Planning	Full	52.99	26.34	54.67	18.06
	w/o INFERRULES	42.54 (-10.45)	25.56 (-0.78)	48.18 (-6.49)	16.67 (-1.39)
	w/o WRAPSTEP	26.12 (-26.87)	16.48 (-9.86)	52.87 (-1.80)	16.67 (-1.39)

Table 15: Ablation study on the components of ALIGN. Values represent task success rates (%) or scores. For ablated conditions (w/o INFERRULES, w/o WRAPSTEP), performance changes from the 'Full' are shown in parentheses.

	WALL-E / AutoManual	APO methods	ALIGN
What is optimized	Agent-side rules or manual	The agent's prompt	The environment's information and feedback functions (f_{info}, f_{obs})
Artifact	Natural-language knowledge injected into the acting agent	Prompt text	Executable interface code (INFERRULES and WRAPSTEP)
Verification	Rules induced from collected trajectories	Held-out scoring of candidate prompts	Controlled experiments executed in the environment before any change is accepted
Runtime feedback seen by the agent	Raw observations, unchanged	Raw observations, unchanged	Rewritten observations (e.g., "Nothing happens" $\rightarrow$ the violated precondition)

Table 16: Positioning of ALIGN against agent-side rule learning (WALL-E, AutoManual) and automatic prompt optimization (APO) methods.

work.

D.3 Content Audit of Injected Observations

Domain	Mechanic	Borderline	Task-specific	Total
ALFWorld	116	6	0	122
WebShop	20	0	0	20
ScienceWorld	71	6	2	79
M^{3}ToolEval	72	0	19†	91
Total	279	12	21	312

Table 17: Content audit of all agent-facing strings injected by the 20 final interfaces. †The 19 M^{3}ToolEval strings instantiate about three distinct patterns.

Interfaces are generated exclusively from training tasks and held fixed during testing (Section 4.1), and the w/o and w/ ALIGN conditions share an identical protocol, so no test-set information enters generation. The stricter question, specific to our own claim that the interface should reveal capability rather than substitute for the reasoning a benchmark measures, is whether the injected content clarifies environment mechanics or over-guides the agent. We audited this directly.

Criterion. An injection is *mechanic-level* iff it is (i) a deterministic function of the current state and the attempted action, i.e., information the environment computed but failed to express, and (ii) free of task-goal content: it never points to the target, spells out a solution procedure, or evaluates answers. Strings that contain only generic hints naming no task entity (e.g., exploration nudges) but are not strictly determined by the state and action are marked *borderline*; strings that violate (ii) are marked *task-specific guidance*. We extracted all 312 agent-facing strings from the 20 final interfaces (four benchmarks $\times$ five agent methods) and labeled each one; the audit data and scripts are released with the code.

Findings. Table 17 shows the result. (1) ALFWorld and WebShop, which are behind the headline results, contain no task-specific guidance; the six borderline ALFWorld strings are generic exploration nudges that name no task entity. (2) The flagged minority in ScienceWorld and M^{3}ToolEval splits into two kinds. One kind is correctness feedback (e.g., "Incorrect, try another"): the generated code computes it at run time from task metadata exposed by the environment, verbalizing a signal that the environment already computes internally; we regard these as environment-design repairs rather than violations, consistent with our finding that richer feedback should be prioritized by environment designers. The other kind encodes procedure-level knowledge for a task type (e.g., an ordering of decoding tools, or a recipe for mixing paint). Such rules are learned from training tasks only and generalize as code, so agent-side learners such as manuals or RL policies would legitimately carry the same knowledge; nevertheless, under our stricter claim we count them as over-guidance. (3) The procedure-level cases correlate exactly with the smallest training splits (one or two tasks per task type, Appendix F.1), where the distinction between mechanics and procedures blurs for the Optimizer, and they occur in the two lowest-gain domains.

Aggregate evidence. Two further checks indicate that the gains are not over-guidance in aggregate. The Valid Actions baseline (Table 9) gives strictly more overt help, namely the full set of legal actions at every turn, yet reaches 44.03% against 60.45% for ALIGN; and the trajectory analysis in Appendix D.4 shows failure-to-success conversions with preserved action entropy and increased exploration.

Mitigation. We add a task-agnosticism constraint to the Optimizer prompt (requirement 9 in Appendix F.4), which rejects injections that refer-

ence task-goal entities or spell out full procedures, and we flag the affected interfaces in the released audit data. The interfaces evaluated in this paper were generated before this constraint was introduced.

D.4 Exploration and Strategy Diversity

A natural concern is that explicitly disclosing implicit rules may undermine the agent's autonomous exploration or collapse its strategy diversity. We therefore analyze the complete evaluation trajectories on ALFWorld and ScienceWorld for all five agent methods; as a sanity check, the same parser reproduces the reductions in consecutive invalid actions reported in Table 2. The analysis supports four observations.

(1) Diversity is preserved. Action-type entropy stays in the same band with and without ALIGN (ALFWorld Vanilla 2.62 vs. 2.72 bits; ScienceWorld Vanilla 3.22 vs. 3.42) and is sometimes higher with ALIGN (ALFWorld ReAct 2.53 vs. 2.30). Unique action trigrams are comparable on ALFWorld and higher with ALIGN on ScienceWorld (Vanilla 14.1 vs. 7.6). The w/o ALIGN runs exhibit more raw action types, but the surplus consists largely of malformed actions emitted while stuck in invalid loops, i.e., noise rather than strategy.

(2) Exploration increases. If rule disclosure substituted for exploration, agents would visit less of the environment; the opposite holds. The mean number of distinct "go to" targets per task rises for every method (ALFWorld Vanilla 5.4 vs. 3.4, Self-Consistency 8.6 vs. 3.1, Self-Refine 4.6 vs. 1.0; ScienceWorld Vanilla 2.6 vs. 1.3). WRAPSTEP tells the agent why an action failed, never what to do next.

(3) Gains are failure-to-success conversions, not smoothing. On ALFWorld, 50–79 test tasks per method are converted from failure to success against only 1–7 regressions; on ScienceWorld the corresponding numbers are 22–46 and 0–9. On tasks solved in both conditions, ALFWorld agents use fewer steps with ALIGN (17.7 vs. 34.0 for Vanilla) because the tail of invalid loops disappears, whereas on ScienceWorld they take more valid actions to reach higher partial-credit scores.

(4) Content. Together with the content audit in Appendix D.3, these results indicate that the improvements stem from restored observability rather than from lowered task difficulty.

D.5 Generator Capability Spectrum

The main configuration uses frontier models in every generator role. To characterize how much capability the generator actually needs, we vary the models driving its three roles (Table 18).

A cheap model suffices for analysis and verification. The Analyzer receives past interaction trajectories in its prompt, so it needs neither a strong world model nor prior knowledge of the environment. Replacing both the Analyzer and the experimental-verification model of the Optimizer with GPT-4.1-mini yields interfaces that are no worse than those of the frontier stack: the interface generated with the Qwen2.5-7B-Instruct agent lifts it from 13.43% to 64.93% (vs. 60.45% with the frontier stack), and the interface generated with the GPT-4.1-mini agent lifts GPT-4.1-mini from 28.36% to 64.93% and GPT-4.1 from 73.88% to 93.28%. The demand for a frontier model therefore concentrates in a single step, the code synthesis of the Optimizer.

A fully self-sufficient 7B stack fails. With Qwen2.5-7B-Instruct in every role, generation does not complete. Across three runs with 32k, 65k, and 131k context windows, which rules out a context-length artifact, the 7B Analyzer never completes the verify-then-hand-off protocol: it proposes 290 misalignment hypotheses but completes zero verifications, so the Optimizer is never invoked and no interface is produced. The bottleneck is capability (structured tool use and exact-signature code generation), not context length.

Takeaways. The generator roles have a real capability floor, which we report as is; the floor matters little in practice, since generation is a one-time cost of about USD 9.4 per environment (Appendix E.1), whereas the agent has no floor: the same 7B model that cannot generate an interface is the one the interface lifts from 13.43% to 60.45%.

D.6 Audit of the Action Transducer

Section 3.1 treats the action transducer f_{act} as the identity mapping. To verify that this scope is empirically appropriate rather than a bypass of intent-to-command translation, we extracted all 87 `env.step` call sites from the 20 deployed interfaces using the Python abstract syntax tree and classified how the agent's action reaches the environment (Table 19). Every interface implements f_{info} (20/20) and f_{obs} (20/20), and none performs a semantic remap of the agent's action, whereas the

Analyzer	Verification	Code synthesis	Outcome (ALFWorld, Vanilla agent)
GPT-4.1	GPT-4.1	Gemini 2.5 Pro	Main configuration (Table 1): Qwen2.5-7B-Instruct 13.43 → 60.45.
GPT-4.1-mini	GPT-4.1-mini	Gemini 2.5 Pro	Works: Qwen2.5-7B-Instruct 13.43 → 64.93 (no worse than the frontier stack); GPT-4.1-mini 28.36 → 64.93 and GPT-4.1 73.88 → 93.28 (Table 14).
Qwen2.5-7B-Instruct	Qwen2.5-7B-Instruct	Qwen2.5-7B-Instruct	Fails to complete generation: across three runs with 32k/65k/131k context, the Analyzer never completes the verify-then-hand-off protocol (290 hypotheses, 0 completed verifications), so the Optimizer is never invoked and no interface is produced.

Table 18: Generator capability spectrum: the models driving the Analyzer, the experimental verification, and the interface code synthesis of the Optimizer, together with the resulting outcome.

How the agent's action reaches the environment	Count
Passed verbatim (execute, then rewrite the observation)	65
Read-only probe or navigation sub-steps added around it	17
Intent-preserving surface normalization (same verb and target)	5
Semantic remap to a different action ($f_{\text{act}} \neq$ identity)	0
Total `env.step` call sites in 20 interfaces	**87**

Table 19: AST audit of all `env.step` call sites in the 20 deployed interfaces.

human-designed interface in Appendix C.2 does ("converting put to move"). In these environments, $f_{\text{act}} =$ identity is thus the empirically correct scope. Environments with a genuine intent-to-command gap, where high-level goals must be compiled into low-level primitives, are the natural extension, and the formulation in Section 3.1 already reserves this slot.

E Token Consumption Analysis

The average token consumption per iteration in the main experiment described in Section 4.1 is shown in Table 20.

Due to the "Experimental Verification" setup, the Analyzer and Optimizer need to interact with the environment multiple times, and all previous interaction histories are included as new prompt inputs to the LLM in each round of interaction. Additionally, when the Optimizer identifies that the generated interface is imperfect, it needs to refine the previously generated interface and conduct experimental verification again, leading to increased token consumption. However, as LLM capabilities continue to improve and hallucination issues decrease, this cost will gradually reduce. Furthermore, it is worth noting that:

- The INFERRULES wrapper and WRAPSTEP wrapper are implemented as python logic code, which does not involve calls to models or agents, therefore not incurring additional token consumption. On the contrary, as demonstrated in our experiments in Section 4.3, **using ALIGN-generated interfaces can help agents reduce repetitive meaningless actions, thereby reducing the number of LLM calls and decreasing token consumption** compared to not using ALIGN-generated interfaces.
- Except when the Optimizer generates interface codes requiring the cutting edge LLMs (such as Gemini 2.5 Pro), weaker and more cost-effective LLMs (such as GPT-4.1-mini) can be used at other times, which will significantly reduce the operational costs of ALIGN. This is validated empirically in Appendix D.5.
- ALIGN-generated interfaces can generalize to different agent architectures and base LLMs. This means that for each environment, using the ALIGN method to generate an interface only once can bring performance improvements to different agents, regardless of agent version updates. This also means that **the cost of interface generation is a one-time expense**, rather than requiring the generation of new interfaces for each task execution. Therefore, from an amortization perspective, the method's cost becomes increasingly economical as the environment is utilized more frequently, with the one-time interface design cost being distributed across multiple uses and becoming proportionally smaller with increased usage.

		ALFWorld	ScienceWorld	WebShop	M^{3}ToolEval
Analyzer	Input Token (M)	0.2770	0.4333	0.1783	0.1094
	Output Token (M)	0.0040	0.0036	0.0048	0.0016
	Total Token (M)	0.2809	0.4370	0.1831	0.1109
Optimizer	Input Token (M)	0.2619	0.2288	0.0669	0.1100
	Output Token (M)	0.0087	0.0172	0.0040	0.0118
	Total Token (M)	0.2706	0.2460	0.0709	0.1217
Total	Total Token (M)	0.5515	0.6830	0.2540	0.2326

Table 20: The average token consumption per iteration in the main experiment described in Section 4.1.

E.1 Cost Accounting

We quantify the cost of ALIGN from the evaluation logs and Table 20.

Generation cost. The 0.55M tokens per iteration on ALFWorld correspond to about USD 1.2 at list prices, so the full $K=8$ iterations cost about USD 9.4, and all four benchmarks together about USD 30. This is a one-time cost per environment: the generated wrapper is model-free Python that is reused by every subsequent agent method, base model, and run.

Inference savings. The interface also reduces the steady-state cost of interaction. From the evaluation logs, agents with ALIGN use 58% of the baseline's input tokens and 43% of its LLM calls per task; the mean per-task output drops from 353 to 142 tokens and the number of calls from 71.2 to 30.6. The interface thus shrinks the interaction footprint rather than inflating it.

Amortization. Over the reuse in this paper (about 3,350 task-runs across agent methods and transferred backbones), generation costs about USD 0.003 per task-run; with a paid-API agent, it is recovered from inference savings alone after about 120 task-runs. For reference, a single 134-task evaluation sweep of GPT-4.1 on ALFWorld (about USD 25) already exceeds the entire generation cost.

Fair comparison with closed-source base models. Table 21 compares Qwen2.5-7B-Instruct equipped with the ALIGN-generated interface against closed-source models used directly as the base model. (i) Qwen2.5-7B-Instruct with ALIGN more than doubles the success rate of the comparably affordable GPT-4.1-mini used as base (60.45% vs. 28.36%), with no recurring API cost. (ii) The choice is not either/or: the same interface applied to the closed-source base yields 93.28%.

E.2 Placement of Strong-Model Capability

ALIGN is itself a collaboration scheme between a strong closed-source model and an open-source small model, with a deliberately chosen structure: the strong model works offline, once per environment, and its output is compiled into a reusable model-free artifact instead of being consulted online. Two natural alternative placements of the same capability can be contrasted with it. (1) *Online per-task teacher guidance* bills the strong model on every task, so the cost grows linearly with usage and the assistance is not reusable, whereas the cost of ALIGN is constant per environment (about USD 0.003 per task-run amortized) and its artifact transfers across five agent architectures and four model families (Table 3 and Tables 10–13). (2) *Strong-model-built agent-side scaffolds* are compared in Appendix C.2: AgentSquare configured with GPT-4.1 plus planning, reasoning, and memory modules reaches at best 86.57% on ALFWorld, against 93.28% for a vanilla agent with the ALIGN-generated interface (Table 8), and interfaces hand-crafted by Ph.D. students reach 47.01% against 60.45% for ALIGN under the same Qwen2.5-7B-Instruct base (Table 9). The question is therefore not whether to add a strong model but where to spend its capability; one-time, verified, environment-side interface synthesis is empirically the highest-leverage placement among those we tested, and a matched-budget comparison against online-teacher schemes is left for future work.

F Implementation Details

F.1 Benchmarks Task Splits

The task splits of benchmarks we use are as follows:

(1) ALFWorld (Shridhar et al., 2021): We adhere to the original dataset partitioning presented in the paper, wherein the tasks from the "eval_out_of_distribution" category are used as the

Configuration	SR (%)	One-time cost	Marginal cost per task
Qwen2.5-7B-Instruct (local), w/o ALIGN	13.43	–	Local GPU only
Qwen2.5-7B-Instruct (local), w/ ALIGN	60.45	≈USD 9.4	Local GPU only; 58% of the baseline's input tokens and 43% of its LLM calls
GPT-4.1-mini as base, w/o ALIGN	28.36	–	Per-task API cost, recurring
GPT-4.1 as base, w/o ALIGN	73.88	–	≈USD 0.18 per task, recurring
GPT-4.1 as base, w/ ALIGN	93.28	≈USD 9.4 (same interface)	≈58% of the row above

Table 21: Cost comparison between an open-source base model equipped with the ALIGN-generated interface and closed-source models used directly as the base model, on ALFWorld with the Vanilla agent. Costs are computed from the evaluation logs and Table 20 at list prices.

test set, and the "train" category is designated as the training set. In each iteration, we randomly select three tasks from the training set of each task type to serve as the training data for the agent's interaction.

(2) ScienceWorld (Wang et al., 2022):We follow the original partitioning of the train and test sets as described in the paper. For efficiency reasons, during testing, we select at most the first five tasks from the 30 available task types for evaluation. In each iteration, we randomly select one task from the training set of each task type to be used as the training data for the agent's interaction.

(3) WebShop (Yao et al., 2022): In alignment with the setup of Yao et al. (2023), we use tasks with IDs ranging from 0 to 49 (50 tasks in total) as the test set, and tasks with IDs from 50 to 199 (150 tasks in total) as the training set. In each iteration, we randomly select 20 tasks from the training set to serve as the training data for the agent's interaction.

(4) M^{3}ToolEval (Wang et al., 2024b): Since M^{3}ToolEval does not provide a distinct training set division, we select two tasks from each task type in the original dataset as the training set, with the remaining tasks used as the test set. In each iteration, the entire training set is utilized for the agent's interaction.

F.2 Hyperparameter and Experimental Setting

For all the agents, we deploy them uniformly using vllm (Kwon et al., 2023) across 8 Nvidia A100 80GB GPUs, with the inference temperature set to 0.0. The models utilized contain Qwen2.5-7B-Instruct (Team, 2024), Qwen2.5-14B-Instruct (Team, 2024), Llama3.1-8B-Instruct (Meta, 2025a) and Llama3.3-70B-Instruct (Meta, 2025b).

In ALIGN, we use Gemini 2.5 Pro (gemini-2.5-pro-exp-03-25)(Google, 2025) for Optimizer to generate new interface, with the temperature set to 0.2. For other scenarios requiring the use of an LLM, we employ GPT-4.1 (gpt-4.1-2025-04-14)(OpenAI, 2025). We set $K = 8$ during experiments.

F.3 Tools for Experimental Verification

In order to implement the experimental verification process mentioned in Section 3.3, we have encapsulated the following tools for Analyzer and Optimizer to interact with the interface-wrapped environment:

(1) `init_simulator(task_id, interface)`: Initializes an experimental task, specifying the task ID and the interface code.

(2) `reset_simulator()`: Resets the experimental task.

(3) `run_task()`: Runs the current task until completion, returning the interaction trajectory.

(4) `exec_agent_action(agent_action)`: Executes a specific action and returns the enhanced observation after the interface processing.

(5) `get_agent_action()`: Based on the current trajectory, returns the next action to be issued by the agent.

(6) `change_obs(obs)`: Modifies the observation of the previous action execution.

F.4 Prompt Templates

We present the prompt template of the Analyzer and Optimizer for ALFWorld. For the prompt templates of other benchmarks, please refer to the supplemental materials. For the WebShop and M^{3}ToolEval environments, no "Gold Action and Observation Sequence" is provided. Requirement 9

https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct
https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct

of the Optimizer prompt for interface generation (the task-agnosticism constraint) was added after the content audit in Appendix D.3.

Analyzer Prompt Template of Misalignment Analysis

User message:
In modern benchmarks evaluating LLM Agent reasoning capabilities, human designers create an Environment with a set of rules defining how tasks are accomplished. These rules, referred to as the Environment's World Model, specify the sequence of actions required to achieve specific outcomes. For example, the Environment's World Model might dictate that certain actions (e.g., operating on a receptacle) can only be performed after prerequisite actions (e.g., moving to the receptacle).

Meanwhile, the Agent operates based on its own World Model, which it constructs by interpreting the task and environment prompts. The Agent first determines its high-level reasoning intent—its understanding of what needs to be done—and then selects actions according to its internal World Model. However, because the Environment's World Model is manually crafted and may not be fully conveyed through prompts, the Agent's World Model might differ, leading to unexpected behavior. For instance, the Agent might choose an action that aligns with its intent but violates the Environment's rules, or it might misinterpret feedback due to insufficient information from the Environment.

We define a misalignment between the Environment's World Model and the Agent's World Model as a situation where:
- The Environment provides feedback that does not sufficiently clarify its World Model, leaving the Agent unable to adjust its understanding of the rules.

Your task is to analyze the logs from a recent task to determine whether such a misalignment occurred, preventing a fair assessment of the Agent's capabilities. And this misalignment has not been fixed by current 'WrapStep' function. Your analysis will guide us in addressing this issue moving forward.

Experimental Environment Evaluation Template

```python
{{ experimental_template }}
```

In this template, the function 'InferRules' is used to define the environment rules. The function 'WrapStep' handles post-processing of the Agent's actions (e.g., splitting them into multiple steps, performing pre-checks, returning more detailed feedback, etc.). This function should not interfere with the Agent's own reasoning. There current implementation is as follows:

```python
{{ Interface }}
```

Environment Logs

```txt
{{ logs }}
```

Here, each 'Observation' is the feedback returned to the Agent after it executes an action.

Gold Action and Observation Sequence

```txt
{{ gold_action_obs_sequence }}
```

Environment Logics and Misalignment Analyzed in the Previous Steps

{{ environment_logics }}

Your Task

Determine whether, during this task, there was a misalignment between the Environment's World Model and the Agent's World Model that hindered a fair assessment of the Agent's capabilities. Choose exactly one of the following outputs:

If there is NO misalignment (i.e., the Agent's failures stem from its own errors or limitations, not a mismatch with the Environment's World Model), output:
<analysis_result> No Misalignment </analysis_result>

If there IS a misalignment (i.e., the Environment's World Model conflicts with the Agent's World Model), output:
<analysis_result> Found Misalignment </analysis_result>
<environment_logic_and_misalignments> the new environment rules and misalignments identified by you, which have not been fixed by current 'WrapStep' function. </environment_logic_and_misalignments>

The format of the environment logic and misalignment is as follows:
```txt
### Analysis Result 1
Analysis Task ID: xxx
Agent Action Type: xxx # The type of action the Agent attempted to perform, such as "examine", "move object to receptacle", etc.
Agent Action Case: xxx # The specific action the Agent attempted to perform.
Agent High-Level Reasoning Intent: xxx # The Agent's high-level reasoning intent, which may be a general description of the action it was trying to perform.
Environment World Model Rule: xxx # The rule from the Environment's World Model that don't align the Agent's World Model.
Sufficient Environment Feedback: xxx # to offer the Agent adequate information to bridge gaps in understanding the environment's world model. such as "The environment should provide 'xxx' feedback when the Agent attempts to operate on a receptacle without first going to it."
Type: "Bug of current WrapStep function" or "Need to add new logic in the WrapStep function"

### Analysis Result 2
...
```

Note: You should not generate duplicate misalignment analysis results as the ones already provided in the 'Environment Logics and Misalignment Analyzed in the Previous Steps' section.

Analyzer Prompt Template of Experimental Verification

User message:
Now you should conduct simulation experiments in the simulator to verify that the environment rules you hypothesized and Misalignment you identified truly exists. You must perform sufficient experiments to confirm or refute your suspicion.

Here are the operations you can use:

1. init_simulator(task_id: str)
- Initializes a new simulator for the specified 'task_id'.
- 'task_id' must be in the format 'int-int' where the first int $\in$ [0, 5].
- The different task types are mapped as follows:

0: 'pick_and_place',
1: 'pick_clean_and_place',
2: 'pick_heat_and_place',

3: 'pick_cool_and_place',
4: 'look_at_or_examine_in_light',
5: 'pick_two_obj_and_place'

- All subsequent operations occur within this initialized simulator.

2. reset_simulator()
- Resets the current simulator to its initial state.

3. execute_agent_action(agent_action: str)
- Executes an agent action using the 'WrapStep' function.

4. change_last_action_observation(obs: str)
- Updates the last observation returned by the simulator to the specified 'obs'.
- This is useful for simulating the agent's next action in a different environment feedback context.

5. get_next_agent_action()
- Retrieves the next action that the real Agent would perform under the current simulation conditions.
- Note: The Agent's choice of the next action is based on the current environment state, including the outcomes of any previous 'step()' or 'get_next_agent_action()' call, along with the latest observations.

If you believe you have reached a conclusion from your experiments, provide it in this format:

<thought> Your reasoning here </thought>
<environment_logic_and_misalignments> the new environment rules and misalignments identified by you, which have not been fixed by current 'WrapStep' function. </environment_logic_and_misalignments>

The format of the environment logic and misalignment is as follows:
"'txt
Analysis Result 1
Analysis Task ID: xxx
Agent Action Type: xxx # The type of action the Agent attempted to perform, such as "examine", "move object to receptacle", etc.
Agent Action Case: xxx # The specific action the Agent attempted to perform.
Agent High-Level Reasoning Intent: xxx # The Agent's high-level reasoning intent, which may be a general description of the action it was trying to perform.
Environment World Model Rule: xxx # The rule from the Environment's World Model that don't align the Agent's World Model.
Sufficient Environment Feedback: xxx # to offer the Agent adequate information to bridge gaps in understanding the environment's world model. such as "The environment should provide 'xxx' feedback when the Agent attempts to operate on a receptacle without first going to it."
Type: "Bug of current WrapStep function" or "Need to add new logic in the WrapStep function"

Analysis Result 2
...
"'

If you need to carry out more operations in the simulator, respond in the following format, specifying exactly one operation per turn:

<thought> Your reasoning here, you should consider all hypotheses if the simulation result is not as expected </thought>
<action> The single operation you wish to perform (e.g., init_simulator(task_id="x-y"), step(action="x"), execute_agent_action(agent_action="x"), etc.) </action>

Note:
You should verify the correctness of the following, step by step, through your experiments:
1. environment_rules: Use 'execute_agent_action' to confirm that the environment rules you hypothesized are indeed correct, and current 'WrapStep' function is not sufficient.
2. agent_intent_description: Obtain

the Agent's intended behavior (e.g., via 'get_next_agent_action') and simulate it by using 'WrapStep' to confirm whether it aligns with your description.
3. identified_misalignment: Through chaning the environment feedback, you can verify whether the misalignment you identified is indeed correct and the environment feedback you hypothesized is indeed sufficient. You can use 'WrapStep' to simulate the agent's action, then use 'change_last_action_observation' to change the environment feedback, and finally use 'get_next_agent_action' to check whether the agent can correctly identify the next action.

Analyzer Prompt Template of Reranking Misalignments Analysis (Ablation Study)

User message:
In modern benchmarks evaluating LLM Agent reasoning capabilities, human designers create an Environment with a set of rules defining how tasks are accomplished. These rules, referred to as the Environment's World Model, specify the sequence of actions required to achieve specific outcomes. For example, the Environment's World Model might dictate that certain actions (e.g., operating on a receptacle) can only be performed after prerequisite actions (e.g., moving to the receptacle).

Meanwhile, the Agent operates based on its own World Model, which it constructs by interpreting the task and environment prompts. The Agent first determines its high-level reasoning intent—its understanding of what needs to be done—and then selects actions according to its internal World Model. However, because the Environment's World Model is manually crafted and may not be fully conveyed through prompts, the Agent's World Model might differ, leading to unexpected behavior. For instance, the Agent might choose an action that aligns with its intent but violates the Environment's rules, or it might misinterpret feedback due to insufficient information from the Environment.

We define a misalignment between the Environment's World Model and the Agent's World Model as a situation where:
- The Environment provides feedback that does not sufficiently clarify its World Model, leaving the Agent unable to adjust its understanding of the rules.

Now other human experts have analyzed the logs from a recent task and identified some potential misalignments. Your task is to review these misalignments and choose the most appropriate one.

Experimental Environment Evaluation Template

```python
{{ experimental_template }}
```

In this template, the function 'InferRules' is used to define the environment rules. The function 'WrapStep' handles post-processing of the Agent's actions (e.g., splitting them into multiple steps, performing pre-checks, returning more detailed feedback, etc.). This function should not interfere with the Agent's own reasoning. There current implementation is as follows:

```python
{{ Interface }}
```

Environment Logs

```txt
{{ logs }}
```

Here, each 'Observation' is the feedback returned to the Agent after it executes an action.

Gold Action and Observation Sequence

```txt
{{ gold_action_obs_sequence }}
```

Environment Logics and Misalignment Analyzed in the Previous Steps

{{ environment_logics }} Note: These logics may not be accurate. They are the environment rules that were previously hypothesized and may contain errors.

Your Task

Choose the most appropriate misalignment analyzed by human experts from the list below:

{{ new_environment_logics }}

You should respond in format as follows:

```
<review> Your review of each expert output one by one </review>
<expert_id> id of the selected expert output, only the number </expert_id>
```

Optimizer Prompt Template of Interface Generation

User message:
In modern benchmarks evaluating LLM Agent reasoning capabilities, human designers create an Environment with a set of rules defining how tasks are accomplished. These rules, referred to as the Environment's World Model, specify the sequence of actions required to achieve specific outcomes. For example, the Environment's World Model might dictate that certain actions (e.g., operating on a receptacle) can only be performed after prerequisite actions (e.g., moving to the receptacle).

Meanwhile, the Agent operates based on its own World Model, which it constructs by interpreting the task and environment prompts. The Agent first determines its high-level reasoning intent—its understanding of what needs to be done—and then selects actions according to its internal World Model. However, because the Environment's World Model is manually crafted and may not be fully conveyed through prompts, the Agent's World Model might differ, leading to unexpected behavior. For instance, the Agent might choose an action that aligns with its intent but violates the Environment's rules, or it might misinterpret feedback due to insufficient information from the Environment.

We define a misalignment between the Environment's World Model and the Agent's World Model as a situation where:
- The Environment provides feedback that does not sufficiently clarify its World Model, leaving the Agent unable to adjust its understanding of the rules.

Your task is to refine the environment's behavior based on the misalignment identified by the AnalysisAgent, ensuring the Agent's true intentions are executed and its reasoning capabilities are fairly assessed.

Experimental Environment Evaluation Template

```python
{{ experimental_template }}
```

In this template, the function ‘InferRules‘ is used to define the environment rules. The function ‘WrapStep‘ handles post-processing of the Agent’s actions (e.g., splitting them into multiple steps, performing pre-checks, returning more detailed feedback, etc.). This function should not interfere with the Agent’s own reasoning. There current implementation is as follows:

“‘python
{{ WrapStep }}
“‘

_____
_____

Environment Logics and Misalignment Analyzed by AnalysisAgent Previously

{{ last_environment_logics }}

_____
_____

New Environment Logics and Misalignment Analyzed by AnalysisAgent

{{ new_environment_logics }}

_____
_____

Your Task

Based on the misalignments identified by the AnalysisAgent, you need to refine and enhance the ‘InferRules‘ function and ‘WrapStep‘ function to align the Environment’s World Model with the Agent’s actions and provide clearer feedback. Your output should present the new versions of these functions, ensuring the Agent’s high-level reasoning intent is preserved.
Please ensure you follow these requirements:

1. **Function Signature**
The function signature must be:
“‘python
def InferRules(init_obs, task)
- init_obs: str, the initial observation from the environment, containing all receptacles.
- task: str, the task description.

def WrapStep(env, init_obs, task, agent_action: str, logger)
“‘

2. **Return Values**
The ‘InferRules‘ function’s return value must be a string that describes the environment rules.

The ‘WrapStep‘ function’s return value must be three items:
“‘python
obs: str, reward: bool, done: bool
“‘

3. **‘env.step‘ Usage**
The only permitted usage pattern for ‘env.step‘ is:
“‘python
obs, reward, done, info = env.step([agent_action])
obs, reward, done = obs[0], info[’won’][0], done[0]
“‘
No alternative usage forms are allowed. Each call to env.step causes an irreversible change to the environment state; actions must therefore be chosen carefully.

4. **Package Imports**
You may import other packages if necessary, but you must include all imports in your code.

5. **Multiple Calls and Conditional Returns**
You are free to call ‘env.step‘ multiple times or return different ‘obs‘ depending on ‘agent_action‘ or the outcomes of these calls.

6. **You can use logger.debug**
You can use ‘logger.debug‘ to log any information you find useful. The logging will be captured and returned to you in the future for further analysis.

7. Do not modify any aspects not explicitly identified by the AnalysisAgent in the "New Environment Logics and Misalignment Analyzed by AnalysisAgent" section.

8. You must use the following approach when addressing the identified misalignment:
- For each action defined in environment, provide clear, informative, and sufficient feedback from the environment whenever an invalid action is attempted, guiding the Agent toward understanding and adhering to the environment's rules.

9. **Task-Agnosticism Constraint**
Every string returned by 'InferRules' or injected into 'obs' by 'WrapStep' must describe environment mechanics only. It must be a deterministic function of the current environment state and the attempted action, and it must not reference task-goal entities, spell out a solution procedure for a task type, or evaluate whether the Agent's answer is correct. Injections that violate this constraint will be rejected during experimental verification.

10. **Output Format**
You must provide the output strictly in the following format:
<thought>YOUR_THOUGHT_PROCESS_HERE<
<code>YOUR_CODE_HERE</code>

Please ensure your final answer follows these guidelines so that we can accurately bridge the misalignment and allow the environment to execute the Agent's true intentions.

Optimizer Prompt Template of Experimental Verification

User message:
Now you should conduct simulation experiments in the simulator to verify if the 'InferRules' and 'WrapStep' function you provided is correct for the new environment logics and misalignment analyzed by the AnalysisAgent.

You must perform sufficient experiments to confirm or refute your suspicion. Here are the operations you can use:

1. init_simulator(task_id: str)
- Initializes a new simulator for the specified 'task_id'.
- 'task_id' must be in the format 'int-int' where the first int $\in$ [0, 5].
- The different task types are mapped as follows:

0: 'pick_and_place',
1: 'pick_clean_and_place',
2: 'pick_heat_and_place',
3: 'pick_cool_and_place',
4: 'look_at_or_examine_in_light',
5: 'pick_two_obj_and_place'

- All subsequent operations occur within this initialized simulator.

2. reset_simulator()
- Resets the current simulator to its initial state.

3. execute_agent_action(agent_action: str)
- Executes an agent action using the 'WrapStep' function you generated.

4. change_last_action_observation(obs: str)
- Updates the last observation returned by the simulator to the specified 'obs'.
- This is useful for simulating the agent's next action in a different environment feedback context.

5. get_next_agent_action()
- Retrieves the next action that the real Agent would perform under the current simulation conditions.
- Note: The Agent's choice of the next action is based on the current environment state, including the outcomes of any previous 'step()' or 'get_next_agent_action()' call, along with the latest observations.

6. run_task(task_id: str)
- Runs the entire task in the simulator and returns the running log.
- After running the whole task, you need to call ‘init_simulator‘ or ‘reset_simulator‘ to reinitialize the simulator for further operations.

If you believe you have reached a conclusion from your experiments, provide it in this format:

<thought> Your reasoning here </thought>
<if_need_refine> True/False </if_need_refine>
<refine_strategy> Your strategy for refining the WrapStep function, if if_need_refine is True </refine_strategy>

If you need to carry out more operations in the simulator, respond in the following format, specifying exactly one operation per turn:

<thought> Your reasoning here, you should consider all hypotheses if the simulation result is not as expected </thought>
<action> The single operation you wish to perform (e.g., init_simulator(task_id="xy"), step(action="x"), execute_agent_action(agent_action="x"), etc.) </action>

Optimizer Prompt Template of Reranking Interface Generation (Ablation Study)

User message:
In modern benchmarks evaluating LLM Agent reasoning capabilities, human designers create an Environment with a set of rules defining how tasks are accomplished. These rules, referred to as the Environment’s World Model, specify the sequence of actions required to achieve specific outcomes. For example, the Environment’s World Model might dictate that certain actions (e.g., operating on a receptacle) can only be performed after prerequisite actions (e.g., moving to the receptacle).

Meanwhile, the Agent operates based on its own World Model, which it constructs by interpreting the task and environment prompts. The Agent first determines its high-level reasoning intent—its understanding of what needs to be done—and then selects actions according to its internal World Model. However, because the Environment’s World Model is manually crafted and may not be fully conveyed through prompts, the Agent’s World Model might differ, leading to unexpected behavior. For instance, the Agent might choose an action that aligns with its intent but violates the Environment’s rules, or it might misinterpret feedback due to insufficient information from the Environment.

We define a misalignment between the Environment’s World Model and the Agent’s World Model as a situation where:
- The Environment provides feedback that does not sufficiently clarify its World Model, leaving the Agent unable to adjust its understanding of the rules.

Now other human experts have generated a set of code patches to address the misalignment between the Environment’s World Model and the Agent’s World Model. Your task is to evaluate these patches and select the best one.

Experimental Environment Evaluation Template

“‘python
{{ experimental_template }}
“‘

In this template, the function ‘InferRules‘ is used to define the environment rules. The function ‘WrapStep‘ handles post-processing of the Agent’s actions (e.g., splitting them into multiple steps,

performing pre-checks, returning more detailed feedback, etc.). This function should not interfere with the Agent's own reasoning. There current implementation is as follows:

```python
{{ WrapStep }}
```

Environment Logics and Misalignment Analyzed by AnalysisAgent Previously

{{ last_environment_logics }}

New Environment Logics and Misalignment Analyzed by AnalysisAgent

{{ new_environment_logics }}

Your Task

Choose the best code from the following options to address the misalignment between the Environment's World Model and the Agent's World Model:

{{ code_patches }}

You should respond in format as follows:

```
<review> Your review of each code one by one </review>
<code_id> id of the selected result, only the number </code_id>
```

We present the prompt template of the Vanilla agent in ALFWorld to illustrate the usage of the INFERRULES. For the prompt templates of other agent methods and benchmarks, please refer to the supplemental materials.

Vanilla Agent Prompt Template in ALFWorld

System message:
You are an AI assistant solving tasks in a household environment. Your goal is to break down complex tasks into simple steps and plan your actions accordingly.

Action Space

In this environment, you have a set of high-level actions at your disposal, each corresponding to a typical household activity. These actions are:

- look: look around your current location
- inventory: check your current inventory
- go to (receptacle): move to a receptacle
- open (receptacle): open a receptacle
- close (receptacle): close a receptacle
- take (object) from (receptacle): take an object from a receptacle
- move (object) to (receptacle): place an object in or on a receptacle
- examine (something): examine a receptacle or an object
- use (object): use an object
- heat (object) with (receptacle): heat an object using a receptacle
- clean (object) with (receptacle): clean an object using a receptacle
- cool (object) with (receptacle): cool an object using a receptacle
- slice (object) with (object): slice an object using a sharp object

Although each action may internally consist of multiple embodied steps (e.g., walking to the sink, turning a knob, etc.), from your perspective you need only provide one high-level action at a time.

Instructions

Single Action per Turn
At each step, you must respond with exactly one action (i.e., the next "thought"). Use the format:
ACTION [object/receptacle specifier]

ACTION [object/receptacle specifier]
For example:
take apple from table
or
go to kitchen.

Environment Feedback
After you provide your single action, the environment will automatically execute it and return the resulting observation. You then decide on your next action based on the updated state.

Reasoning (Chain of Thought)
You may use hidden reasoning to figure out the best next step. However, only output the single action that represents your decision. Do not reveal your entire chain of thought.

Continue Until Task Completion
You will iterate this process—receiving the environment's feedback, deciding on the next action, and outputting a single action—until the task is finished.

Environment Rule

{InferRules(init_obs, task)}

User message:
Task

{initial_obs}

Begin by examining the environment or taking any initial steps you find relevant. Remember, provide only one action each time.