DUALFRONT: WHAT MUST HOLD AND WHAT MUST COME FIRST FOR LONG-HORIZON LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon interactive tasks require LLM agents to satisfy long-term objectives while adhering to immediate execution constraints. Existing paradigms enhance decision-making through planning, memory management, or state tracking. However, these mechanisms often fail to explicitly couple goal fulfillment with execution readiness. Incoherent state updates lead to *state drift*—a misalignment where the agent's context diverges from the current task state, resulting in actions that either deviate from remaining requirements or violate current preconditions. We propose **DualFront**, a state-adaptation framework that synchronizes agent reasoning through dual dynamic frontiers driven by interaction evidence. The *goal frontier* identifies a goal condition to establish or verify (*What Must Hold*), while the *execution frontier* identifies a prerequisite state to address first (*What Must Come First*). By jointly updating these frontiers from shared evidence, DualFront ensures that execution preparation remains structurally aligned with the current task state. As a plug-and-play guidance layer, DualFront is compatible with various agent architectures, including ReAct, Plan-and-Execute, and Reflexion. Evaluations across three interactive environments show that DualFront improves macro-averaged success rates by up to 15.78 percentage points while reducing average LLM calls per episode by 9.47%. Trajectory analysis further reveals a reduction in execution mismatch rates. These results demonstrate that DualFront improves task completion and call efficiency across agent paradigms, while better aligning actions with current execution conditions.

1 INTRODUCTION

Large language model (LLM) agents have demonstrated remarkable potential in navigating complex, interactive environments (Liu et al., 2024a; Zhou et al., 2024b). For long-horizon tasks, success hinges on the agent's ability to maintain a consistent internal representation of task progress while dynamically adapting to environmental feedback (Agashe et al., 2025). Ideally, an agent must keep its decision-making context strictly aligned with both the **remaining task requirements** (what remains to be done) and the **current execution conditions** (what is currently possible).

However, as interaction depth increases, LLM agents frequently suffer from **state drift**—a phenomenon where the agent's internal decision context diverges from the environment state (Rozanov & Rei, 2025). We categorize this loss of alignment into two critical dimensions: (1) **Goal-state misalignment**, where the agent performs redundant or irrelevant actions that deviate from remaining requirements; and (2) **Execution-state misalignment**, where the agent attempts actions that violate latent physical or logical preconditions. As illustrated in Figure 1(a), an agent may redundantly reheat an item that is already hot (goal misalignment) or attempt to place an object before reaching the destination (execution misalignment). Our trajectory audit across three environments reveals that such drift is not merely incidental but systemic: accepted actions need not advance remaining task requirements, while rejection rates are higher later in execution and during longer periods without progress (Section 2). These observations motivate two ongoing assessments: what remains to be satisfied, and whether the prerequisites for advancing those requirements currently hold.

Existing research addresses these challenges through various lenses, yet coordinating goal satisfaction with execution readiness remains an open problem. Planning-based methods (Sun et al., 2023; Erdogan et al., 2025) attempt to revise sequences based on feedback but often fail to verify

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

if previously satisfied goals remain valid. Memory-centric approaches (Hu et al., 2025; Zhou et al., 2026) compress history to aid decisions but lack explicit mechanisms to reconcile historical success with current execution constraints. While state-tracking frameworks (Yoneda et al., 2024; Anokhin et al., 2025) maintain environmental knowledge, they do not inherently link these updates to the *prioritization* of remaining goals. Crucially, existing solutions often incur significant overhead, either requiring auxiliary LLM calls for coordination (Kim & Hwang, 2025) or necessitating rigid, specialized workflows (Liu et al., 2025; Li et al., 2026) that limit cross-paradigm generalizability.

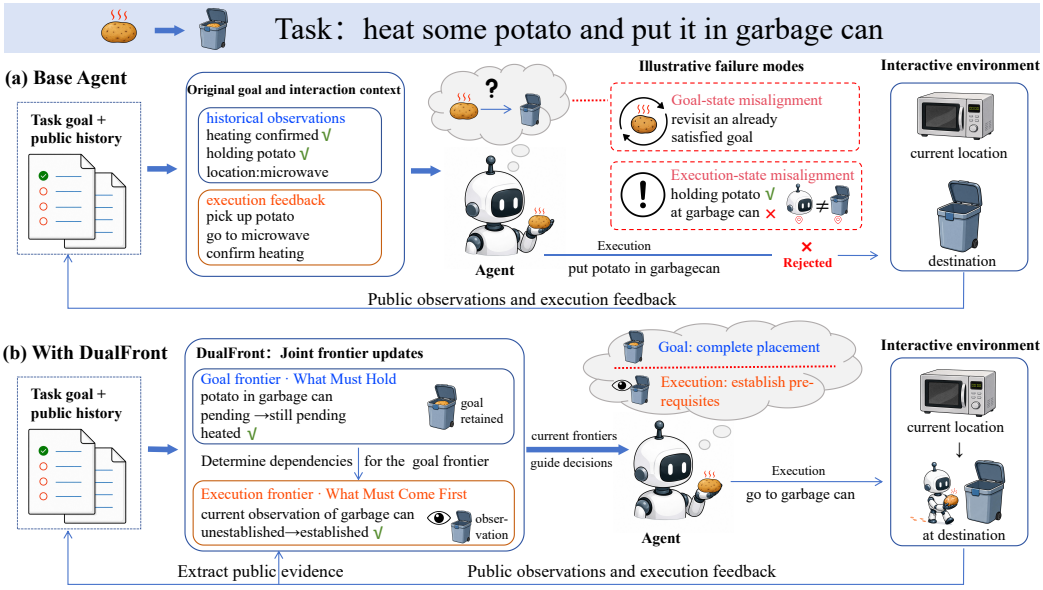


Figure 1: State drift in a base agent and joint frontier updates in DualFront.

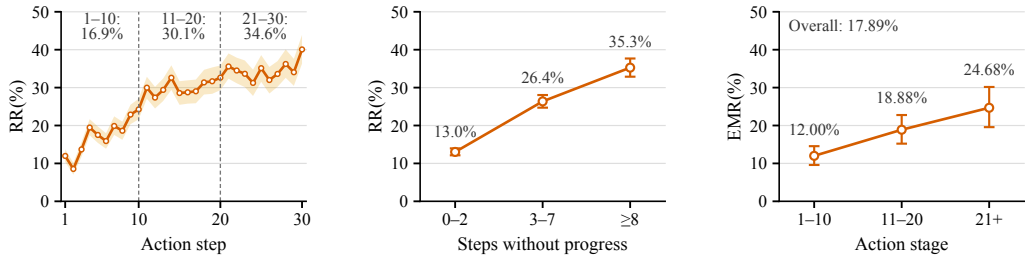
To bridge this gap, an effective state-adaptation mechanism must satisfy three core design objectives: **(1) Non-monotonic Goal Assessment:** It must dynamically identify unmet goals, including those invalidated by subsequent environment changes, to prevent redundant or missed actions. **(2) Coupled Goal-Execution Logic:** It must prioritize execution prerequisites based on specific remaining goals, ensuring the agent addresses "what must come first" to satisfy "what must hold." **(3) Efficiency and Orthogonality:** It should provide state guidance without auxiliary LLM calls or intrusive modifications to the base agent's core decision loop.

We propose **DualFront**, a framework that synchronizes task requirements with execution readiness through **dual dynamic frontiers** updated via public interaction evidence. The *Goal Frontier* isolates the next goal condition to establish or verify, while the *Execution Frontier* identifies the immediate prerequisite state required to bridge the gap. As shown in Figure 1(b), arrival and observation at the garbage can satisfy the prerequisite, shifting the focus to placement while retaining the goal. This process requires zero additional LLM calls and is inherently compatible with diverse agent paradigms, including Act, ReAct, and Reflexion. Experimental results across three interactive environments demonstrate that DualFront significantly enhances agent performance. DualFront with Qwen2.5-32B-Instruct improves macro-averaged success and progress rates by 10.57 and 7.24 percentage points over the corresponding base agents. Despite the added guidance, it reduces pooled mean input tokens per episode by 7.23%. Mean LLM calls per episode decrease by 9.47%, while the pooled action rejection rate falls by 3.37 percentage points. Experiments with another backbone also show performance gains.

2 Motivation

To investigate the origin of agent failure in long-horizon interactions, we audit trajectories generated by Qwen2.5-32B-Instruct across ALFWorld (Shridhar et al., 2021), ScienceWorld (Wang et al., 2022), and PDDL (Silver & Chitnis, 2020) under four base-agent paradigms. We use **rejection**

rate (RR) to measure environment-rejected actions and **execution mismatch rate** (EMR) to assess grounding and precondition errors. **Accepted state revisit rate** (ASRR) quantifies returns to previously visited states through accepted task-operation actions, supporting analysis of repeated behavior in relation to remaining requirements. Formal definitions and statistical protocols are provided in Appendix A.1.



(a) RR by action step (b) RR by steps without progress (c) PDDL EMR by stage

Figure 2: Baseline action rejection and execution mismatch. (a,b) Cross-environment RR from 1,136 trajectories; (c) PDDL EMR from 240 trajectories. Shading and error bars show 95% confidence intervals from 2,000 bootstrap resamples.

Goal-state misalignment. Our audit reveals a significant *efficiency-validity gap*: in PDDL tasks, the ASRR reaches 21.52%, indicating that over one-fifth of the actions merely cycle through known states. While revisits are occasionally necessary for exploration, trajectory analysis reveals that some repeated behaviors do not serve unmet goals. For instance, in the *Barman* task, an agent may repeatedly pick up and put down a glass after its filling goal is satisfied (Appendix A.4). Crucially, when an environmental change invalidates a previously satisfied condition, the agent often fails to recognize the regression, continuing to execute redundant operations until budget exhaustion. ***Observation 1: Execution validity does not imply goal relevance.*** This suggests that current LLM agents lack a *non-monotonic* mechanism to reassess goal satisfaction dynamically based on cumulative feedback.

Execution-state misalignment. We observe that the agent's decision-making quality deteriorates as the interaction depth increases. The pooled RR roughly doubles from 16.9% (steps 1–10) to 34.6% (steps 21+), as shown in Figure 2a. And RR rises sharply during periods of stagnation, increasing from 13.0% after 0–2 steps without progress to 35.3% after at least 8 such steps (Figure 2b). This trend is mirrored by the EMR, which climbs from 12.00% to 24.68% (Figure 2c). These patterns persist even in sophisticated memory-based paradigms, as shown by our reproduction of HiAgent (Hu et al., 2025) (Appendix A.2). ***Observation 2: Precondition violations scale with trajectory depth and progress stagnation.*** These patterns are consistent with execution-state misalignment, highlighting the need to reassess current execution conditions as interaction proceeds.

Design implications. The observed misalignments highlight a critical coordination problem. Identifying a remaining goal is insufficient if its execution prerequisites are ignored. Conversely, execution preparation must serve the remaining task requirements. This motivates the design of DualFront, which jointly maintains: (i) a goal frontier to select a goal condition to establish or verify; and (ii) an execution frontier to identify a prerequisite state to address first, based on the selected goal and current environmental feedback. By synchronizing these dual frontiers, we aim to mitigate state drift with minimal inference overhead.

3 METHOD

Given a task goal x and public interaction history τ_t, DualFront jointly updates the goal and execution frontiers to identify a state to establish or verify. Figure 3 illustrates the overall process. At initialization, DualFront compiles x into a goal program $\mathcal{G}$. At each step, it updates goal satisfaction and selects the goal frontier $g_t^\star$. It then instantiates a local state-dependency graph for this goal using environment-level schemas and the current public state, and selects the execution frontier $p_t^\star$. The execution focus can advance while the goal is retained. When the goal changes, its dependencies are rebuilt. A renderer converts the selected state focus into guidance $\mathcal{I}_t$ for the base agent, which generates environment actions. The resulting public feedback drives the next update.

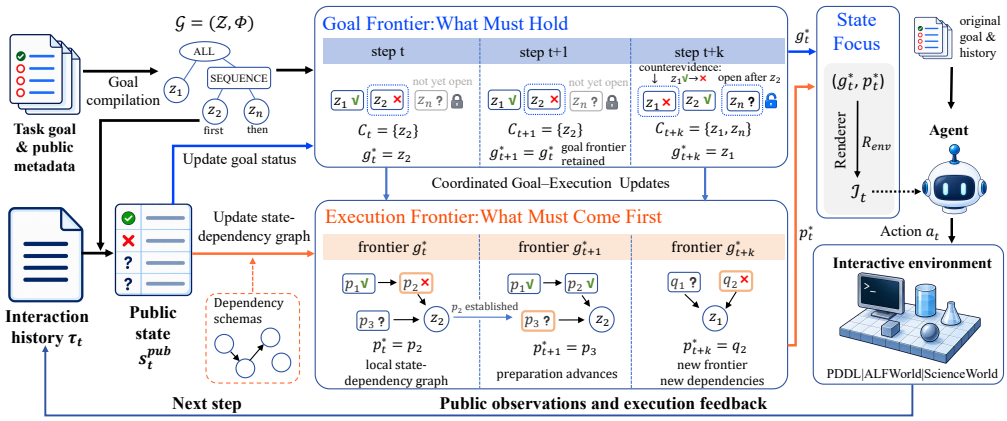


Figure 3: DualFront overview and coordinated frontier updates. From t to $t+1$, the goal frontier remains at z_2, while the execution frontier advances from p_2 to p_3 as prerequisites are satisfied. At $t+k$, counterevidence invalidates z_1, shifting the goal frontier to z_1 and triggering reconstruction of its local dependency graph. Here, z_i denote goal atoms, and p_i and q_i denote prerequisite states. Checkmarks, crosses, and question marks indicate satisfied, unsatisfied, and unknown states.

3.1 Task Representation and Goal Compilation

For tasks in different interactive environments, DualFront incorporates the corresponding public metadata m^{pub}, such as domain names, and compiles the task goal x into a unified goal program.

$$C_{\text{env}}(x, m^{\text{pub}}) \longrightarrow \mathcal{G} = (\mathcal{Z}, \Phi). \tag{1}$$

Here, C_{env} implements a shared compilation interface using environment-specific parsing rules, and $\mathcal{Z} = \{z_1, \ldots, z_n\}$ contains the goal atoms to track. Each atom specifies a requirement to verify:

$$z_i = \langle \psi_i(\cdot; \theta_i), \rho_i, \kappa_i, \mu_i \rangle. \tag{2}$$

$\psi_i(\cdot; \theta_i)$ specifies the required relation, with parameters θ_i. For example, "ball1 is at roomb" is represented by $\texttt{AT}(\cdot; \texttt{ball1}, \texttt{roomb})$, where $\theta_i = (\texttt{ball1}, \texttt{roomb})$. The placeholder $\cdot$ takes the public state maintained during execution (Section 3.2). ρ_i records the source goal text, and κ_i specifies the required instance count, defaulting to one. μ_i distinguishes two satisfaction semantics: *current-state* and *past-satisfaction*. The former requires continued assessment and allows later counterevidence to invalidate satisfaction. The latter retains a completion record once the requirement is satisfied. An object being in a specified container follows the former semantics, whereas completing a specified observation follows the latter. Fixed at initialization, these attributes govern subsequent state updates. Φ specifies the logical composition of goal atoms through a recursive grammar with five operators:

$$\begin{aligned} \Phi \ ::= z_i \mid \text{ALL}(\Phi_1, \ldots, \Phi_m) \mid \text{ANY}(\Phi_1, \ldots, \Phi_m) \\ \mid \text{SEQUENCE}(\Phi_1, \ldots, \Phi_m) \mid \text{COUNT}_k(z_i) \mid \text{CONDITIONAL}(b, \Phi^+, \Phi^-). \end{aligned} \tag{3}$$

Here, Φ_j, Φ^+, and Φ^- are goal subexpressions generated by the same grammar, each representing a single atom or a nested composition. m denotes the number of child subexpressions. ALL requires every child subexpression to be satisfied, whereas ANY requires at least one. SEQUENCE requires ordered confirmation and opens only the first unsatisfied subexpression for consideration. $\text{COUNT}_k(z_i)$ requires at least $k = \kappa_i$ instances to satisfy the condition specified by z_i. CONDITIONAL selects between Φ^+ and Φ^- according to a public condition b; neither branch is opened while that condition remains unresolved. Compilation details are provided in Appendix B.1.

3.2 Dynamic Goal Frontier

As interaction proceeds, DualFront uses public evidence to reassess goal satisfaction and determine which goals may currently be advanced. It selects a goal that remains to be satisfied or verified from $\mathcal{G}$, answering *What Must Hold*. Environments provide observations and execution feedback in

different forms. An environment adapter Γ^s_{env} organizes information relevant to $\mathcal{G}$ from the public history τ_t into a public state $s_t^{\text{pub}} = \Gamma^s_{\text{env}}(\mathcal{G}, \tau_t)$. For each atom z_i, the evidence interface Γ^z_{env} checks the relation $\psi_i(s_t^{\text{pub}}; \theta_i)$ defined in Section 3.1 and produces atom evidence:

$$\Delta_{i,t} = \Gamma^z_{\text{env}}(s_t^{\text{pub}}, z_i) = \langle v_{i,t}, n_{i,t}, e_{i,t} \rangle. \tag{4}$$

Here, $v_{i,t} \in \{+, -, \varnothing\}$ denotes supporting evidence, counterevidence, or no new judgment; $n_{i,t}$ is the applicable instance count, and $e_{i,t}$ records the public basis for the judgment. The absence of new evidence is not treated as counterevidence. Satisfaction updates also account for the atom's count requirement and satisfaction semantics. For each z_i, DualFront maintains $\sigma_t(z_i) \in \{\mathsf{T}, \mathsf{F}, \mathsf{U}\}$, denoting satisfied, unsatisfied, and unknown, respectively. All atoms are initially unknown. Let $\Delta_t = \{\Delta_{i,t}\}_{i=1}^n$. Satisfaction states are updated as follows (Appendix B.2):

$$\boldsymbol{\sigma}_t = \text{Update}\left(\mathcal{G}, \boldsymbol{\sigma}_{t-1}, \Delta_t, s_t^{\text{pub}}\right). \tag{5}$$

The update first checks whether the evidence is applicable under the goal logic, taking the previous satisfaction states into account. It then checks whether the supported instance count reaches κ_i. Finally, μ_i determines whether later counterevidence can revoke prior satisfaction. Without a new applicable judgment, the previous state is retained. After the update, the goal candidate set contains atoms that are logically open and not yet satisfied:

$$\mathcal{C}_t = \{z_i \in \mathcal{Z} \mid \sigma_t(z_i) \neq \mathsf{T},\ z_i \text{ is currently open under } \Phi\}. \tag{6}$$

When $\mathcal{C}_t \neq \varnothing$, DualFront retains the previous goal if it remains a candidate. Otherwise, it selects $g_t^\star$ by prioritizing unsatisfied atoms with explicit evidence, followed by other unsatisfied atoms and then unknown atoms. The selected $g_t^\star$ anchors dependency expansion for the execution frontier.

3.3 Dynamic Execution Frontier

The goal frontier $g_t^\star$ identifies a remaining requirement but does not establish whether its execution conditions are met. DualFront therefore uses an execution frontier to identify a prerequisite state to establish or verify, answering *What Must Come First*. Given $g_t^\star$, DualFront combines the goal relation with the current public state s_t^{pub} to construct a local dependency graph (Appendix B.2):

$$D_t = \Gamma^D_{\text{env}}(g_t^\star, s_t^{\text{pub}}) = (V_t \cup \{g_t^\star\}, E_t). \tag{7}$$

V_t contains prerequisite-state nodes for the current goal, each with status $\delta_t(v_i) \in \{\mathsf{T}, \mathsf{F}, \mathsf{U}\}$. An edge $(u, v) \in E_t$ indicates that u must hold before v can be advanced. The mapping Γ^D_{env} combines state relations in an environment's dependency schema with current entity bindings. Schemas define local partial orders over states based on public task semantics. They contain no concrete actions, search paths, or complete execution plans. Graph updates are performed programmatically. After each interaction, the graph is re-instantiated from the current goal and latest public feedback, updating entity bindings and node statuses.

Starting from the immediate predecessors of $g_t^\star$, DualFront traverses the updated graph backward. If $\delta_t(v_i) = \mathsf{T}$, traversal stops along that branch. Otherwise, v_i is added to the pending set $\mathcal{R}_t$, and its predecessors are examined. The execution candidate set then consists of pending nodes whose immediate predecessors are all satisfied:

$$\mathcal{F}_t = \{p \in \mathcal{R}_t \mid \forall u :\ (u, p) \in E_t \Rightarrow \delta_t(u) = \mathsf{T}\}. \tag{8}$$

When $\mathcal{F}_t \neq \varnothing$, the previous execution node is retained if it remains a candidate. Otherwise, DualFront prioritizes unsatisfied nodes over unknown nodes to select the execution frontier $p_t^\star$.

Each round of public feedback first updates the goal frontier, then its execution dependencies and frontier. Even with an unchanged goal, feedback may alter entity bindings and prerequisite satisfaction, changing the execution candidates. A goal change triggers dependency instantiation for the new goal. Together, the frontiers determine the state focus for the next action.

3.4 State Guidance and Agent Integration

After updating $g_t^\star$ and $p_t^\star$, DualFront represents the current goal and any selected prerequisite as a joint state focus:

$$\xi_t = \begin{cases} \langle g_t^\star, p_t^\star \rangle, & \text{a goal exists and reliable state guidance is available,} \\ \varnothing, & \text{otherwise.} \end{cases} \tag{9}$$

Here, $p_t^\star$ may be empty. When all relevant prerequisites are satisfied, guidance focuses directly on the goal $g_t^\star$. An environment renderer R_{env} converts ξ_t into textual guidance $\mathcal{I}_t = R_{\text{env}}(\xi_t)$. It preserves the goal context and expresses the selected prerequisite or the ready goal using the environment's entity names and conventions, without changing either frontier. When $\xi_t = \varnothing$ or guidance cannot be rendered reliably, $\mathcal{I}_t = \varnothing$ and the base agent retains its original task input. At each action decision where $\mathcal{I}_t \neq \varnothing$, DualFront appends "`Current dynamic subgoal:`$\mathcal{I}_t$" to the original task. This preserves the base agent's action parsing, history organization, and model-call workflow. DualFront makes no additional LLM calls, but guidance adds input tokens. We report this overhead separately. Appendix B.3 provides the online algorithm and an interaction example.

4 Experiments

We evaluate DualFront through five research questions: **RQ1:** Does DualFront improve task success and progress across environments and agent paradigms? **RQ2:** Can it improve efficiency and action reliability without additional LLM calls for state adaptation? **RQ3:** How do goal compilation, the goal frontier, and the execution frontier contribute to performance? **RQ4:** Are performance gains accompanied by fewer execution mismatches and accepted state revisits? **RQ5:** How do DualFront's gains vary with action budget, guidance-injection frequency, and backbone model?

4.1 Experimental Setup

Tasks and Environments. Following AgentBoard (Ma et al., 2024), we evaluate on three text-based interactive environments: **PDDL** (Silver & Chitnis, 2020), covering symbolic planning in Gripper, Blocksworld, Barman, and Tyreworld; **ALFWorld** (Shridhar et al., 2021), covering partially observable household tasks; and **ScienceWorld** (Wang et al., 2022), covering scientific tasks involving navigation, manipulation, measurement, and conditional reasoning.

Baselines and Implementation. Main experiments use Qwen2.5-32B-Instruct (Qwen Team, 2025) with four agent paradigms: Act (Yao et al., 2023), ReAct (Yao et al., 2023), Plan-and-Execute (Wang et al., 2023a), and Reflexion (Shinn et al., 2023), each evaluated with and without DualFront. We use AgentBoard's task instructions and one-shot demonstrations, a 30-step action budget, temperature 0, top-p 1, and seed 0. Implementation and evaluation details appear in Appendix C.1.

Metrics and Aggregation. We report success rate (SR), the fraction of tasks completed within the budget, and progress rate (PR), the mean of the highest task-completion scores across episodes. Costs are measured by environment actions, LLM calls, and input/output tokens. Rejection counts and RR assess action reliability; EMR and ASRR characterize execution mismatches and accepted state revisits (Appendix A.1). Macro Avg. weights environments equally; overall gains also weight agent paradigms equally. SR and PR gains are in percentage points; cost reductions are relative percentages. Further metric definitions and aggregation procedures appear in Appendix C.2.

4.2 Answer 1: Improved Success and Progress

To assess DualFront's effect on task performance, we compare SR and PR with and without DualFront across three environments and four agent paradigms (Table 1). **Overall performance.** DualFront improves SR and PR in all 12 environment–paradigm pairs. With equal weighting across environments and paradigms, SR increases from 41.44% to 52.00% (+10.57 pp), and PR from 63.12% to 70.35% (+7.24 pp). Mean SR gains are 12.08, 11.01, and 8.61 pp on PDDL, ALFWorld, and ScienceWorld, respectively, covering symbolic, household, and scientific tasks. These improvements indicate that DualFront helps agents complete more tasks and achieve higher task progress within the same action budget. **Performance across paradigms.** Mean SR gains across environments are 15.78, 6.80, 6.07, and 13.61 pp for Act, ReAct, Plan-and-Execute, and Reflexion, respectively. These gains show that DualFront benefits agents with different control structures. The smaller gain for Plan-and-Execute may reflect its existing task decomposition and execution-order guidance.

4.3 Answer 2: Improved Efficiency and Action Reliability

Table 2 compares execution costs and action rejections with and without DualFront across all environments and paradigms. **LLM calls and environment actions.** In the pooled results, DualFront

Table 1: Task performance of Baseline and DualFront. Macro Avg. averages over environments; Mean averages over agent paradigms. Bold marks the better result within each pair; underlining marks the best of the eight configurations in each column. Shaded rows represent DualFront.

Method	PDDL		ALFWorld		ScienceWorld		Macro Avg.	
	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑
Act	20.00	44.54	62.69	72.51	36.67	68.60	39.78	61.89
+ DualFront	**35.00**	**56.82**	**80.60**	**86.82**	**51.11**	**72.87**	**55.57**	**72.17**
ReAct	28.33	55.72	73.88	77.61	28.89	55.61	43.70	62.98
+ DualFront	**38.33**	**63.02**	**79.85**	**83.02**	**33.33**	**61.48**	**50.51**	**69.18**
P&E	26.67	53.97	75.37	81.90	31.11	65.77	44.38	67.22
+ DualFront	**33.33**	**56.80**	**83.58**	**86.94**	**34.44**	**66.72**	**50.45**	**70.15**
Reflexion	18.33	47.01	64.18	71.70	31.11	62.42	37.87	60.38
+ DualFront	**35.00**	**59.00**	**76.12**	**81.41**	**43.33**	**69.34**	**51.48**	**69.91**
Mean (Baseline)	23.33	50.31	69.03	75.93	31.94	63.10	41.44	63.12
Mean (+ DualFront)	**35.42**	**58.91**	**80.04**	**84.55**	**40.56**	**67.60**	**52.00**	**70.35**
Δ (pp)	+12.08	+8.60	+11.01	+8.61	+8.61	+4.50	+10.57	+7.24

Table 2: Execution cost and action reliability by environment and agent paradigm. Calls, actions, and tokens are episode averages; k denotes 1,000 tokens. All pools the four paradigms within an environment; Pooled aggregates all episodes rather than weighting environments equally. Bold marks the lower value in each pair. Shaded rows use DualFront.

Env.	Agent	Setting	Mean count/episode		Mean tokens/episode		Action reliability	
			LLM↓	Actions↓	Input (k)↓	Output (k)↓	Rejects↓	RR(%)↓
PDDL	Act	Baseline	27.45	27.45	80.02	0.303	236	14.33
		+ DualFront	**25.55**	**25.55**	**75.49**	**0.293**	**189**	**12.33**
	ReAct	Baseline	29.40	25.93	68.19	0.727	261	16.77
		+ DualFront	**27.98**	**25.02**	**66.52**	**0.595**	**214**	**14.26**
	P&E	Baseline	27.30	26.30	70.53	0.531	219	13.88
		+ DualFront	**26.90**	**25.90**	**70.48**	**0.523**	**177**	**11.39**
	Reflexion	Baseline	27.47	27.47	65.68	0.297	265	16.08
		+ DualFront	**25.70**	**25.70**	**62.58**	**0.281**	**202**	**13.10**
	All	Baseline	27.90	26.79	71.10	0.464	981	15.26
		+ DualFront	**26.53**	**25.54**	**68.77**	**0.423**	**782**	**12.76**
ALFWorld	Act	Baseline	17.12	17.12	20.95	0.150	424	18.48
		+ DualFront	**13.96**	**13.96**	**17.09**	**0.126**	**143**	**7.64**
	ReAct	Baseline	19.04	16.28	20.38	0.542	**229**	**10.49**
		+ DualFront	**16.02**	**14.69**	**17.55**	**0.403**	343	17.42
	P&E	Baseline	15.72	14.45	17.05	0.189	232	11.98
		+ DualFront	**14.50**	**13.43**	**16.16**	**0.170**	**109**	**6.06**
	Reflexion	Baseline	17.84	17.46	19.33	0.165	317	13.55
		+ DualFront	**15.24**	**14.92**	**16.75**	**0.154**	**219**	**10.96**
	All	Baseline	17.43	16.33	19.43	0.262	1,202	13.74
		+ DualFront	**14.93**	**14.25**	**16.89**	**0.213**	**814**	**10.66**
ScienceWorld	Act	Baseline	23.06	23.06	63.61	0.187	826	39.81
		+ DualFront	**19.22**	**19.22**	**52.79**	**0.158**	**639**	**36.94**
	ReAct	Baseline	24.53	24.50	61.31	**0.179**	1,165	52.83
		+ DualFront	**23.87**	**23.76**	**60.10**	**0.179**	**979**	**45.79**
	P&E	Baseline	26.59	23.43	68.91	0.359	916	43.43
		+ DualFront	**25.79**	**23.22**	**67.42**	**0.322**	**766**	**36.65**
	Reflexion	Baseline	25.10	23.28	61.78	0.227	954	45.54
		+ DualFront	**22.64**	**20.99**	**56.15**	**0.205**	**813**	**43.04**
	All	Baseline	24.82	23.57	63.90	0.238	3,861	45.51
		+ DualFront	**22.88**	**21.80**	**59.11**	**0.216**	**3,197**	**40.74**
Pooled		Baseline	21.98	20.83	44.44	0.297	6,044	25.54
		+ DualFront	**19.90**	**19.03**	**41.23**	**0.258**	**4,793**	**22.17**

improves task success while reducing LLM calls per episode from 21.98 to 19.90 (9.47%) and environment actions from 20.83 to 19.03 (8.65%). These reductions reflect both changes in execution and earlier termination after success. **Token consumption.** We cap guidance text at 640 characters to limit its input cost. The injected block averages 26.07 tokens, including the fixed prefix (Appendix D.1). Although guidance adds tokens, input tokens per episode decrease from 44.44k to 41.23k (7.23%), and output tokens from 0.297k to 0.258k (13.02%). Thus, savings from fewer calls outweigh the added guidance tokens in this evaluation. **Action rejections.** Total rejections decrease from 6,044 to 4,793 (20.70%). RR also falls from 25.54% to 22.17% (-3.37 pp), showing that the reduction is not solely due to fewer actions. Since rejection does not directly measure execution-state misalignment, Section 4.5 examines EMR and ASRR to relate these changes to state adaptation.

4.4 Answer 3: Contributions of Goal Compilation and Both Frontiers

We use incremental ablations to assess the benefits of static structured goals, dynamic goal selection, and goal-specific execution preparation. Across the four PDDL agent paradigms, we compare four conditions (Table 3): **Baseline** uses only the original task goal and public interaction history; **w/o GF & EF** retains goal compilation and provides its output as a fixed structured hint; **w/o EF** updates the goal frontier and selects $g_t^\star$ from public feedback, without expanding execution dependencies; **DualFront** enables both the goal frontier (GF) and execution frontier (EF). Static guidance improves mean SR and PR over Baseline by 2.50 and 1.97 pp, respectively. Adding GF increases mean SR by 2.08 pp but decreases mean PR by 2.86 pp. Adding EF improves mean SR and PR by 7.50 and 9.49 pp over w/o EF, yielding the highest SR and PR in all four paradigms. These results support combining current goal requirements with guidance on prerequisite states to establish or verify.

Table 3: Frontier ablations on PDDL. "w/o" denotes removal of the specified component(s). Mean weights the four paradigms equally. Bold marks the best result. The shaded row is full DualFront.

Variant	Act		ReAct		P&E		Reflexion		Mean	
	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑
Baseline	20.00	44.54	28.33	55.72	26.67	53.97	18.33	47.01	23.33	50.31
w/o GF & EF	21.67	48.09	35.00	59.37	26.67	52.11	20.00	49.57	25.83	52.28
w/o EF	26.67	46.67	31.67	52.47	28.33	50.52	25.00	48.02	27.92	49.42
DualFront	**35.00**	**56.82**	**38.33**	**63.02**	**33.33**	**56.80**	**35.00**	**59.00**	**35.42**	**58.91**

4.5 Answer 4: Fewer Execution Mismatches and Accepted State Revisits

Figure 4 compares EMR and ASRR across the four PDDL configurations in Section 4.4 (Appendix A.3). DualFront reduces EMR from 17.89% to 14.62% and ASRR from 21.52% to 16.68% relative to Baseline. Replacing static hints with dynamic goal guidance lowers the ASRR point estimate but raises EMR from 17.63% to 20.01%. Focusing on the current goal alone is therefore insufficient to improve execution readiness. Adding EF reduces EMR by 5.39 pp, supporting explicit prerequisite guidance and aligning with the completion gains in Section 4.4. On the Barman task from the motivation analysis, public feedback confirms that `shot3` contains its required ingredient. DualFront then selects the `shot6` goal and guides the agent to hold that glass. Once this prerequisite holds, the agent fills `shot6` under the same goal, then advances to the `shot9` goal. DualFront completes the task in eight steps without execution mismatches or accepted state revisits, whereas Baseline fails after 30 steps (Appendix A.4).

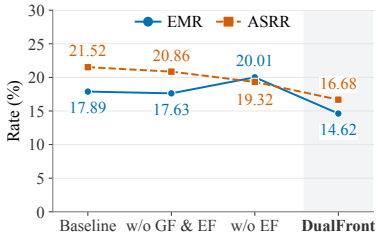


Figure 4: EMR and ASRR on PDDL. Each configuration pools 240 trajectories across four paradigms.

4.6 Answer 5: Effects of Budget, Injection Frequency, and Backbone Model

We vary the action budget and guidance-injection frequency on PDDL with Qwen2.5-32B-Instruct, and evaluate Qwen2.5-14B-Instruct (Qwen Team, 2025) across all three environments to assess sen-

sitivity to the backbone model. **Maximum action budget.** We compare Baseline and per-step DualFront at action budgets $H \in \{5, 10, 15, 20, 25, 30\}$ across four paradigms. Each budget is evaluated through independent runs on the same 60 PDDL tasks. Shorter-budget results are not obtained by truncating 30-step trajectories. Figure 5 shows higher PR for DualFront at every evaluated budget from 10 to 30 steps, with mean gains increasing from 2.88 pp at 10 steps to 8.18 pp at 25 steps. Gains at larger budgets are greater for Act and Reflexion than for P&E. At five steps, PR is similar, with DualFront slightly below Baseline for Act and Reflexion. Thus, the gains are not restricted to the default budget but remain limited when few interactions are allowed.

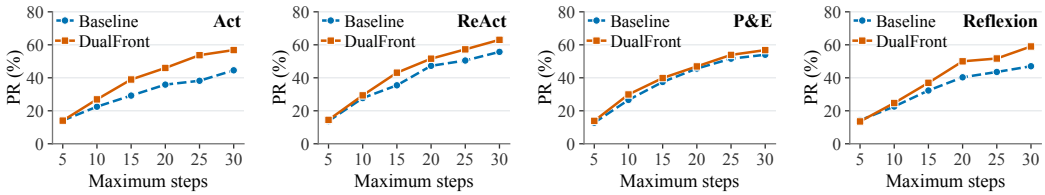


Figure 5: Effect of the maximum environment-action budget on PDDL task progress.

Injection frequency. Among six schedules, combining change-triggered and periodic injection gives the highest mean SR and PR, although the effects vary by agent paradigm (Appendix D.2).

Backbone model. With Qwen2.5-14B-Instruct (Qwen Team, 2025), DualFront improves macro-averaged SR and PR by 9.30 and 8.11 pp, respectively (Appendix D.3).

5 Related Work

LLM agents execute long-horizon tasks through extended sequences of interdependent actions, using observations and interaction history to inform subsequent decisions. Existing approaches organize such behavior through goal decomposition and feedback-driven plan revision (Sun et al., 2023; Wang et al., 2023b; Liu et al., 2024b; Yang et al., 2025b), while search and policy-level reflection further support decision-making (Zhou et al., 2024a; Zhang et al., 2024). Another line of work preserves and reuses information through structured memories and learned memory operations (Xu et al., 2025; Yu et al., 2026), and manages growing interaction histories through context compression and staged memory consolidation (Kang et al., 2026; Fang et al., 2026). Prior experience can also be distilled into context-aware guidelines for subsequent decisions (Fu et al., 2024). Beyond planning and memory, observation–action representations and predictive world models help ground decisions in the environment (Yang et al., 2025a; Chae et al., 2025). Knowledge-based approaches incorporate action constraints, task-transition relations, and prerequisite reasoning into planning and execution (Zhu et al., 2025; Zhao et al., 2025; Liu et al., 2026). Task-state alignment further examines whether task stages, observed evidence, memory, and delegated executors remain consistent during execution (Guo et al., 2026). However, past completion does not guarantee that a goal condition still holds, and selecting a remaining goal does not establish that its prerequisites are currently satisfied. These challenges motivate jointly updating dynamic goal and execution frontiers to keep long-horizon decisions aligned with remaining task requirements and current execution conditions.

6 Conclusion, Limitations, and Future Work

We introduced DualFront, a state-adaptation method that jointly updates goal and execution frontiers from public interaction evidence. By aligning remaining task requirements with execution preparation, DualFront provides state guidance without additional LLM calls or changes to the base agent's decision loop. Experiments across three environments and four agent paradigms show improved success and progress rates, with fewer LLM calls and lower token consumption per episode on average. Our evaluation focuses on benchmark tasks with bounded interaction budgets. Applicability to open-ended and multimodal tasks remains to be validated. Future work will explore how the two frontiers can adapt to open-ended settings where users revise task goals and requirements during execution, supporting more flexible long-horizon interaction.

AI USE STATEMENT

We used generative AI tools to assist with manuscript writing and editing, as well as literature search and discovery. Writing assistance included translation, language polishing, and sentence rephrasing to improve clarity and readability. Literature assistance included identifying relevant studies and helping us read and understand the literature. We also used AI tools to assist with coding and the design of experimental scripts. The authors take full responsibility for the content of the final manuscript and the accuracy of its references.

ETHICS STATEMENT

This study adheres to the ICLR Code of Ethics and does not involve human participants or animal experiments. The experiments use publicly available benchmark environments and datasets and do not involve the collection or use of personally identifiable information.

REPRODUCIBILITY STATEMENT

We have made every effort to ensure the reproducibility of our results. The main text and appendices describe the experimental setup, model configurations, and evaluation procedures in detail. We plan to release the relevant code to facilitate replication and verification of our results.

REFERENCES

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

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. AriGraph: Learning knowledge graph world models with episodic memory for LLM agents. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 12–20, 2025.

Hyungjoo Chae, Namyoung Kim, Kai Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *International Conference on Learning Representations*, volume 2025, pp. 63707–63738, 2025.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-Act: Improving planning of agents for long-horizon tasks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 15419–15462, 2025.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. LightMem: Lightweight and efficient memory-augmented generation. In *International Conference on Learning Representations*, volume 2026, pp. 98706–98729, 2026.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems*, 37: 119919–119948, 2024.

Shuhan Guo, Kun Zhang, Haifei Liu, Xingyu Gao, Yongqi Zhang, Yaqing Wang, and Quanming Yao. ContextFlow: Hierarchical task-state alignment for long-horizon embodied agents. *arXiv preprint arXiv:2605.19314*, 2026.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. HiAgent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32779–32798, 2025.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Minsoo Kim and Seung-won Hwang. CoEx—co-evolving world-model and exploration. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 21629–21651, 2025.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Y Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. In *International Conference on Learning Representations*, volume 2026, pp. 50524–50570, 2026.

Anthony Zhe Liu, Xinhe Wang, Jacob Sansom, Yao Fu, Jongwook Choi, Sungryull Sohn, Jaekyeom Kim, and Honglak Lee. Interactive and expressive code-augmented planning with large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 20330–20354, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024a.

Xiaotian Liu, Armin Toroghi, Jiazhou Liang, David Courtis, Ruiwen Li, Ali Pesaranghader, Jaehong Kim, Tanmana Sadhu, Hyejeong Jeon, and Scott Sanner. Natural language PDDL (NL-PDDL) for open-world goal-oriented commonsense regression planning in embodied AI. In *International Conference on Learning Representations*, 2026.

Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. Reason for future, act for now: A principled architecture for autonomous LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 31186–31261, 2024b.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. *Advances in Neural Information Processing Systems*, 37:74325–74362, 2024.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

Nikolai Rozanov and Marek Rei. StateAct: Enhancing LLM base agents via self-prompting and state-tracking. In *Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025)*, pp. 367–385, 2025.

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Tom Silver and Rohan Chitnis. PDDLGym: Gym environments from PDDL problems. In *Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL), ICAPS 2020*, 2020.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with language models. *Advances in Neural Information Processing Systems*, 36:58202–58245, 2023.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634, 2023a. doi: 10.18653/v1/2023.acl-long.147.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Shawn Ma, and Yitao Liang. Describe, explain, plan and select: interactive planning with LLMs enables open-world multi-task agents. *Advances in Neural Information Processing Systems*, 36:34153–34189, 2023b.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2025.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik A Chaudhari, George Karypis, and Huzefa Rangwala. AgentOccam: A simple yet strong baseline for LLM-based web agents. In *International Conference on Learning Representations*, volume 2025, pp. 97533–97565, 2025a.

Ruihan Yang, Jiangjie Chen, Yikai Zhang, Siyu Yuan, Aili Chen, Kyle Richardson, Yanghua Xiao, and Deqing Yang. SELFGOAL: Your language agents already know how to achieve high-level goals. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 799–819, 2025b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Takuma Yoneda, Jiading Fang, Peng Li, Huanyu Zhang, Tianchong Jiang, Shengjie Lin, Ben Picker, David Yunis, Hongyuan Mei, and Matthew R Walter. Statler: State-maintaining language models for embodied reasoning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15083–15091. IEEE, 2024.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21457–21483, 2026.

Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. Agent-Pro: Learning to evolve via policy-level reflection and optimization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5348–5375, 2024.

Haiyu Zhao, Zhenyu Guo, Chunhong Zhang, Ziyu Zhou, and Zheng Hu. StructuThink: Reasoning with task transition knowledge for autonomous LLM-based agents. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 24489–24506, 2025.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 62138–62160, 2024a.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024b.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, volume 2026, pp. 58413–58438, 2026.

Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, Huajun Chen, and Ningyu Zhang. KnowAgent: Knowledge-augmented planning for LLM-based agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 3709–3732, 2025.

APPENDIX

A MOTIVATION ANALYSIS AND TRAJECTORY REPLAY EVIDENCE

This appendix provides the measurement protocol for the motivation analysis, supplementary rejection statistics from reproduced HiAgent, replay statistics for the dual-frontier mechanism, and a Barman trajectory comparison. All replays use completed experimental trajectories without generating new actions. Complete symbolic states are used only for offline evaluation, not as online input to the base agent or DualFront.

A.1 METRICS AND REPLAY PROTOCOL FOR THE MOTIVATION ANALYSIS

Metric definitions. An *environment action* is a request submitted to the environment that consumes the action budget. Internal reasoning, planning, summarization, and memory retrieval do not count as environment actions. Observations and valid-action queries are *information actions*, such as `look around` and `check valid actions` in PDDL. All other environment actions are *task-operation actions*, whether accepted or rejected. The three metrics are defined as

$$\mathrm{RR} = \frac{N_{\mathrm{rej}}}{N_{\mathrm{all}}}, \qquad \mathrm{EMR} = \frac{N_{\mathrm{ground}} + N_{\mathrm{pre}}}{N_{\mathrm{rec}}}, \qquad \mathrm{ASRR} = \frac{N_{\mathrm{revisit}}}{N_{\mathrm{acc}}}. \tag{10}$$

Rejection rate (RR). RR is the fraction of all environment actions that are rejected. N_{all} counts all actions submitted to the environment, including task-operation and information actions. N_{rej} counts actions explicitly rejected by the environment. We use the acceptance and validity flags in the online logs: an action is rejected if either flag is explicitly false.

Execution mismatch rate (EMR) and accepted state revisit rate (ASRR). EMR measures the fraction of recognizable task-operation actions whose arguments or preconditions fail to meet execution requirements. N_{rec} counts task-operation actions with recognizable schemas. N_{ground} counts actions with missing arguments or invalid object bindings. N_{pre} counts actions with valid bindings but unsatisfied preconditions; these two error categories are disjoint. Information actions and actions with unrecognizable schemas are excluded. ASRR measures the fraction of accepted task-operation actions that return to a previously visited state. N_{acc} counts accepted task-operation actions. N_{revisit} counts those whose successor state equals a complete state previously visited in the same trajectory, including the initial state. Both EMR and ASRR are computed by deterministic offline trajectory replay.

Trajectory coverage. The baseline audit uses existing Qwen2.5-32B-Instruct records under Act, ReAct, Plan-and-Execute, and Reflexion. Each paradigm covers 60 PDDL, 134 ALFWorld, and 90 ScienceWorld tasks, giving 1,136 trajectories across the four paradigms. RR uses all three environments; EMR and ASRR use the 240 PDDL trajectories. The 60 PDDL problems comprise 20 Gripper, 10 Blocksworld, 20 Barman, and 10 Tyreworld problems. Each trajectory has a budget of 30 environment actions. Agent implementations and paired settings are described in Appendix C.1.

Offline trajectory replay. We load the PDDL domain and problem files identified by the domain name and problem index in each log. We reset the environment to the original initial state and check the initial observation. We then execute the recorded actions in order, without LLM calls or action correction or replacement. For action a_t, we read the complete symbolic fact sets before and after execution, S_{t-1} and S_t, and record which goal facts hold. Deterministic procedures make the following judgments: **Execution mismatch.** We first identify the action schema and check argument bindings against the domain's object table. An unrecognizable schema is excluded from EMR. For a recognizable schema, missing arguments or invalid object bindings count as grounding errors. Once arguments are bound, we check preconditions in S_{t-1}. A missing required fact or a present forbidden fact counts as a precondition mismatch, and the relevant facts are recorded. If all conditions hold, no execution mismatch is recorded. **Accepted state revisit.** Each trajectory maintains its own set of visited states, initialized with S_0. After an accepted task-operation action, we compare S_t with all previously visited states. Equal complete fact sets count as a revisit; the visited-state set is then updated. Fact order is ignored. Rejected actions and information actions do not count as revisits, and changes in goal progress do not provide an additional filter.

Replay validation. We compare each replay with its original log, checking action acceptance, highest recorded progress, termination flags, and final success. Acceptance and termination must match exactly. If a mismatch occurs, we record its location and exclude that action and all subsequent actions from valid replay statistics. In addition to these programmatic checks, we manually reviewed representative trajectories against the task goals, original actions and feedback, and replayed states.

Grouping and statistical methods. Execution stages are steps 1–10, 11–20, and 21 onward; only executed actions are counted. Stagnation is the number of consecutive steps without an increase in the highest recorded progress, measured before the current action. We group it into 0–2, 3–7, and at least 8 steps. The 95% confidence intervals in Figure 2 and Table 4 use 2,000 bootstrap resamples with seed 0. We report the 2.5th and 97.5th percentiles. For cross-environment RR, we stratify by environment and paradigm and sample complete trajectories with replacement, preserving each stratum's size. For PDDL EMR and ASRR, we sample 60 problem IDs with replacement from the 60 problems and retain all actions from all four paradigms for each sampled problem. Each resample preserves the original action groups and recomputes rates from pooled counts. Table 4 reports the counts, grouped statistics, and confidence intervals underlying the motivation analysis.

Table 4: PDDL statistics for the motivation analysis. Brackets give 95% confidence intervals for overall metrics. Rates and intervals are percentages. Grouped rows list the denominators and EMR values in the stated order; + means at least.

Metric	Count / denominator	Rate and 95% CI
EMR	981/5,484	17.89% [14.75, 21.23]
ASRR	969/4,503	21.52% [15.47, 28.02]
EMR: steps 1–10 / 11–20 / 21+	2,109 / 1,811 / 1,564	12.00 / 18.88 / 24.68%
EMR: stagnation 0–2 / 3–7 / 8+	2,054 / 1,422 / 2,008	8.28 / 14.21 / 30.33%

A.2 Additional Motivation Evidence from Reproduced HiAgent

Reproduction setup. We extend the rejection-rate analysis to our local reproduction of HiAgent (Hu et al., 2025), using Qwen2.5-32B-Instruct on 134 ALFWorld, 90 ScienceWorld, and 60 PDDL tasks. Each episode has a budget of 30 environment actions, with temperature 0, top-p 1, and seed 0. The resulting 284 trajectories contain 6,369 environment actions. RR follows the definition in Appendix A.1, with rejections identified by the logged action-acceptance flag. Execution stages use steps 1–10, 11–20, and 21 onward. Rates pool rejection and action counts within each group, and HiAgent trajectories are analyzed separately from the four baseline paradigms.

Table 5: Rejection rates in our HiAgent reproduction, pooled across all three environments by execution stage and stagnation duration. Values are descriptive point estimates.

Grouping	Steps	RR (%)	Grouping	Steps	RR (%)
Execution steps	1–10	34.38	Steps without progress	0–2	22.42
	11–20	63.54		3–7	48.46
	21+	72.92		8+	73.36

A.3 Replay Statistics and Paired Comparisons across Four Configurations

Comparison setup. We use the metric definitions, replay procedure, and validation criteria in Appendix A.1 to compare the four ablation configurations in the main text. B0 is Baseline; B1 is w/o GF & EF, which retains goal compilation and supplies static guidance; B2 is w/o EF, which retains the dynamic goal frontier; and B3 is full DualFront. Configurations are paired by problem and agent paradigm. Each configuration includes 240 trajectories from the same 60 PDDL problems across four paradigms. The full set contains 960 trajectories and 25,218 actions, all validated by

replay. Table 6 reports EMR and ASRR for each configuration. We compare adjacent configurations and Baseline versus DualFront on the same task set.

Table 6: Replay metrics and differences across four configurations. Parentheses specify the reference configuration. Δ is the change relative to that reference, in percentage points; negative values indicate reductions. B3 is compared with both B2 and B0.

Configuration (reference)	EMR (%)	ASRR (%)	ΔEMR	ΔASRR
B0	17.89	21.52	—	—
B1 (vs. B0)	17.63	20.86	-0.26	-0.66
B2 (vs. B1)	20.01	19.32	$+2.39$	-1.53
B3 (vs. B2)	14.62	16.68	-5.39	-2.64
B3 (vs. B0)	14.62	16.68	-3.27	-4.83

A.4 Trajectory Comparison

Task and initial conditions. The case is a PDDL Barman task executed by Qwen2.5-32B-Instruct under Act, with global episode ID 48 and domain-specific `problem_index=18`. The task requires three facts to hold simultaneously: g_1 is `shot3 contains ingredient3`, g_2 is `shot6 contains ingredient2`, and g_3 is `shot9 contains ingredient1`. Initially, all three goals are unsatisfied. The relevant glasses are empty, clean, and on the table, and both hands are free. Baseline and DualFront use the same task goal, problem index, and initial observation. Both are independent runs from the same initial conditions, validated by symbolic replay.

Compact coverage of the full trajectories. Table 7 covers all 30 Baseline steps and all eight DualFront steps in nine rows. Consecutive actions are grouped into stages and described by their meaning. The state column records which goal facts hold at the end of each interval, ordered as (g_1, g_2, g_3); 1 means true and 0 means false. Actions and public feedback come from the original logs. Goal facts and revisit labels are verified by offline symbolic replay.

Table 7: Compact Barman trajectory comparison, covering every action interval in both runs. Key events are goal invalidation at step 13 and the failure to restore it during steps 20–30.

Setting	Steps	Actions and key changes	(g_1, g_2, g_3)
Baseline	1–2	Pick up and fill `shot3`, satisfying the first goal.	$(1, 0, 0)$
	3–10	Put down, then repeatedly pick up and put down `shot3`; other goals remain unmet.	$(1, 0, 0)$
	11–12	Another fill is rejected; query valid actions.	$(1, 0, 0)$
	13	**Pour the ingredient into the shaker. The action is accepted but invalidates g_1.**	$\mathbf{(0, 0, 0)}$
	14–19	Put down `shot3`; pick up and fill `shot6`, then `shot9`.	$(0, 1, 1)$
	20–30	**Put down, then repeatedly pick up and put down `shot9`; exhaust the budget without restoring g_1.**	$\mathbf{(0, 1, 1)}$
DualFront	1–2	Pick up and fill `shot3`.	$(1, 0, 0)$
	3–5	Put down `shot3`; pick up and fill `shot6`.	$(1, 1, 0)$
	6–8	Put down `shot6`; pick up and fill `shot9`, completing the task.	$(1, 1, 1)$

Baseline: goal invalidation and missed restoration. After satisfying g_1 at step 2, the agent repeatedly picks up and puts down `shot3` at steps 4–10. The fill attempt at step 11 is rejected because the empty-glass and cleanliness preconditions do not hold. Step 12 only queries available actions and is excluded from execution mismatch and state revisit statistics. At step 13, pouring the ingredient into the shaker invalidates g_1, creating a restoration requirement. The agent then satisfies g_2

and g_3, but repeatedly picks up and puts down `shot9` during steps 21–30. It exhausts its budget without restoring g_1. The misalignment lies in continued manipulation of glasses whose goals are already satisfied and the missed restoration of an unmet goal.

DualFront: goal switching and execution preparation. After feedback at step 2 confirms g_1, the goal frontier switches to g_2 and the execution frontier selects holding `shot6` as the next prerequisite. The agent puts down `shot3` and picks up `shot6`. Once holding is confirmed, it continues toward g_2 and fills the glass at step 5. It then switches to g_3, first prepares to hold the glass, and fills it at step 8 to satisfy all goals. This sequence illustrates switching after goal satisfaction and preparing execution conditions while retaining a goal.

Baseline fails after 30 steps, comprising 29 task-operation actions and one information query. It has one execution mismatch and 17 accepted state revisits, giving EMR $1/29$ and ASRR $17/28$. All eight DualFront actions are accepted task operations. They satisfy all goals with no execution mismatches or state revisits. This comparison illustrates how DualFront shifts attention to remaining goals after satisfaction and updates prerequisite guidance while retaining the current goal.

B Environment Adaptation, Evidence Extraction, and Dual-Frontier Implementation

Environment adapters use public task formats and semantic conventions to extract goal relations and parameters. They determine satisfaction semantics, count requirements, and logical composition, then compile these requirements into a unified goal program. The adapter interfaces handle differences in task expressions and feedback across environments, supporting coordinated updates of goal and execution states. This appendix describes the compilation interface, goal-program structure, public evidence extraction, and local execution dependencies. It also provides the full algorithm and an interaction example.

B.1 Compilation Interface and Goal-Program Structure

Compilation interface. C_{env} consists of environment-specific parsers and shared structural validation. Its unified interface receives the environment identifier, goal text x, and public metadata m^{pub}. The environment identifier selects the parser, while available domain or task-family information provides context for interpreting the goal. The parser identifies goal requirements, extracts relation types, parameters, counts, and conditions, and constructs atom records and their logical composition. PDDL uses public goal conditions; ALFWorld uses public task text; ScienceWorld combines task text with task-family identifiers. Each parser produces a goal program $\mathcal{G} = (\mathcal{Z}, \Phi)$, where $\mathcal{Z}$ stores the individual requirements and Φ specifies their logical composition.

Internal structure of the goal program. The implementation stores $\mathcal{G}$ as atom records and a recursive expression tree. Each atom record defines a requirement to verify; tree leaves reference atoms through `atom_id`, and internal nodes store logical operators and their child expressions. An atom's semantic definition is stored separately from its position in the expression tree. The compilation result also records the environment, task family, original goal, and parsing status for adaptation and validation. Table 8 maps the main fields to the notation in the main text.

The predicate field in Table 8 identifies the relation type of each goal requirement. We define these types by environment: for example, `AT` denotes an object being at a specified location, `CONTAINS` denotes a container holding a required ingredient, `PLACED` denotes an object meeting a specified placement requirement, and `OBSERVED_VALUE` denotes a measurement obtained for a specified property of an object. κ_i and μ_i are fixed at compilation. κ_i defaults to 1 and is set to 2 when two distinct instances are required. μ_i distinguishes *current-state* from *past-satisfaction*: the former requires continued assessment and allows later counterevidence to invalidate satisfaction; the latter retains a completion record once the requirement is satisfied. For conditional tasks, compilation records the comparison rules and branch requirements. Public measurements obtained during interaction determine the condition b.

Table 8: Main fields of the goal program. Atom identifiers link expressions, evidence, and state records. Satisfaction states are maintained separately during interaction.

Field	Notation	Meaning
`atom_id`	ID of z_i	Links expression leaves, evidence, and satisfaction records; the identifier itself does not encode satisfaction.
`predicate`, `arguments`	ψ_i, θ_i	Relation type and parameters that jointly specify the requirement to verify.
`source_text`	ρ_i	Goal clause or task text from which the requirement is extracted.
`required_count`	κ_i	Required number of qualifying instances, with a default of one.
`modality`	μ_i	*Current-state* or *past-satisfaction* semantics, determining whether later counterevidence can revoke confirmed satisfaction.
`root`, `children`	Φ and its children	Nested structure for conjunction, disjunction, sequence, count, and conditional composition.
`guard`	b	Public-quantity references, comparison operators, thresholds, and units used to select a conditional branch.
`attributes`	Additional metadata	Task-specific constraints and annotations.

B.2 Public Evidence Interfaces and Execution Dependencies

Public evidence interfaces. Γ^{s}_{env} organizes public information, whereas Γ^{z}_{env} extracts evidence for individual atoms. They are implemented by programs for observation parsing, entity tracking, and goal-relation verification, with modules selected by environment. Here, an environment is the interactive environment associated with a benchmark. PDDL provides symbolic planning environments. ALFWorld provides household environments for finding, processing, and placing objects. ScienceWorld provides scientific environments for tasks such as measurement and changes in the state of matter. Γ^{s}_{env} uses rule-based text parsing, including string matching and regular expressions, to extract task-relevant facts and events from public observations and feedback. It incrementally maintains these records to form s^{pub}_{t}. Γ^{z}_{env} matches public records against each atom's parameters according to its relation type, distinguishing supporting evidence, counterevidence, and cases where no new judgment can be made. It produces atom evidence $\Delta_{i,t} = \langle v_{i,t}, n_{i,t}, e_{i,t} \rangle$ for subsequent state updates.

Satisfaction-state updates. Update first uses Φ, the previous satisfaction states, and the current public conditions to determine which atoms may receive their first positive confirmation in the current round. Satisfied subexpressions are skipped when determining openness; ALL and ANY recursively expand unsatisfied subexpressions, SEQUENCE expands only the first child whose status is not T, COUNT expands its associated atom, and CONDITIONAL expands only the selected branch, with neither branch expanded while the condition remains unknown. These restrictions prevent premature confirmation of requirements that are not yet open; previously satisfied atoms remain subject to subsequent evidence checks.

Applicable atom evidence $\Delta_{i,t} = \langle v_{i,t}, n_{i,t}, e_{i,t} \rangle$ is then converted into satisfaction states. For supporting evidence $v_{i,t} = +$, if $n_{i,t} \geq \kappa_i$, the state is set to T and the requirement is recorded as having been satisfied; otherwise, the state is set to F. For counterevidence $v_{i,t} = -$, if μ_i specifies *current-state* semantics, the state is set to F, revoking any prior satisfaction. If it specifies *past-satisfaction* semantics and a confirmed completion record exists, the previous state is retained; otherwise, the state is set to F. When $v_{i,t} = \varnothing$ or the evidence is inapplicable, the previous state remains unchanged. After the updates, logical openness is reevaluated using the new satisfaction states to form the goal candidate set $\mathcal{C}_t$. An atom whose satisfaction is revoked by counterevidence reenters the candidate set if it is logically open.

Local dependency graph construction. Γ^{D}_{env} is implemented by an environment-specific dependency expander. The expander organizes reusable dependency schemas by goal relation, describing relevant prerequisite states, parameter-binding conventions, and dependencies between states and

from states to the goal. At runtime, the expander selects a schema for the environment and current goal relation. It binds entities using goal parameters and s_t^{pub} to produce the local dependency graph D_t. Node statuses are determined through relation matching and condition checks: each node's condition and bound parameters are checked against public records. A node is marked T when its condition holds, F when the records suffice to determine that it does not hold, and U when necessary information is missing. After each round of feedback, the update procedure re-instantiates the graph for the current goal, updating entity bindings and node statuses. When the goal changes, it performs this process for the new goal.

B.3 Full Online Procedure and Interaction Example

Algorithm 1 gives the coordinated update procedure for both frontiers. Each iteration first updates goal satisfaction from public feedback and selects the goal frontier. It then updates the dependency graph and execution frontier for that goal. While the goal remains active, the execution frontier advances as prerequisite states change. When the goal changes, execution dependencies are instantiated for the new goal. The two frontiers jointly determine the state focus for the base agent's action; the resulting public feedback drives the next iteration. The operator $\oplus$ denotes appending guidance to the original task input; empty guidance leaves that input unchanged.

Algorithm 1 DualFront: Coordinated Online Frontier Updates

Require: Environment env, public goal x, public metadata m^{pub}, base agent π_ω, action budget H
Ensure: Environment-action trajectory $(a_0, a_1, \ldots)$

$\mathcal{G} = (\mathcal{Z}, \Phi) \leftarrow C_{\text{env}}(x, m^{\text{pub}})$
Receive o_0; set $\tau_0 \leftarrow (o_0)$, $t \leftarrow 0$; initialize counts and evidence records
$\boldsymbol{\sigma}_{-1} \leftarrow (\mathsf{U}, \ldots, \mathsf{U})$, $g^\star_{-1}, p^\star_{-1} \leftarrow \varnothing$
while the environment has not terminated and $t < H$ **do**
  $s_t^{\text{pub}} \leftarrow \Gamma^s_{\text{env}}(\mathcal{G}, \tau_t)$
  $\Delta_t \leftarrow \{\Gamma^z_{\text{env}}(s_t^{\text{pub}}, z_i)\}_{i=1}^n$
  $\boldsymbol{\sigma}_t \leftarrow \mathrm{Update}(\mathcal{G}, \boldsymbol{\sigma}_{t-1}, \Delta_t, s_t^{\text{pub}})$
  Form $\mathcal{C}_t$ from logically open atoms with $\sigma_t(z_i) \neq \mathsf{T}$
  $g_t^\star, p_t^\star, \xi_t, \mathcal{I}_t \leftarrow \varnothing$
  if $\mathcal{C}_t \neq \varnothing$ **then**
    if $g^\star_{t-1} \in \mathcal{C}_t$ **then**
      $g_t^\star \leftarrow g^\star_{t-1}$
    else
      Select $g_t^\star \in \mathcal{C}_t$, prioritizing F with recorded evidence, then other F, then U
    end if
  else if the goal logic is satisfied but public completion confirmation is pending **then**
    Select the most recently updated atom supporting completion as verification goal $g_t^\star$
  end if
  if $g_t^\star \neq \varnothing$ **then**
    $D_t \leftarrow \Gamma^D_{\text{env}}(g_t^\star, s_t^{\text{pub}})$; update node statuses δ_t
    if the relevant dependencies are reliably determined and D_t is valid **then**
      Form $\mathcal{R}_t$ by traversing backward from $g_t^\star$'s immediate predecessors; stop at satisfied nodes
      $\mathcal{F}_t \leftarrow \{p \in \mathcal{R}_t \mid \forall u :\ (u, p) \in E_t \Rightarrow \delta_t(u) = \mathsf{T}\}$
      if $\mathcal{F}_t \neq \varnothing$ **then**
        $p_t^\star \leftarrow p^\star_{t-1}$ if $p^\star_{t-1} \in \mathcal{F}_t$; otherwise select $p_t^\star$, preferring F to U
        $\xi_t \leftarrow \langle g_t^\star, p_t^\star \rangle$
      else if no prerequisites are required (confirmed), or the goal has immediate predecessors and all are satisfied **then**
        $\xi_t \leftarrow \langle g_t^\star, \varnothing \rangle$
      end if
    end if
  end if
  $\mathcal{I}_t \leftarrow R_{\text{env}}(\xi_t)$; set $\mathcal{I}_t \leftarrow \varnothing$ if no reliable guidance can be rendered
  $a_t \sim \pi_\omega(\cdot \mid x \oplus \mathcal{I}_t, \tau_t)$
  Execute a_t and receive o_{t+1}; $\tau_{t+1} \leftarrow (\tau_t, a_t, o_{t+1})$
  $t \leftarrow t + 1$
end while

Example over successive interactions. The example comes from a PDDL Gripper run with Qwen2.5-32B-Instruct under ReAct, with domain-specific `problem_index=0`. The task requires placing `ball1` through `ball4` in `roomb`. These four location requirements correspond to $z_1, \ldots, z_4$, combined as $\Phi = \mathrm{ALL}(z_1, z_2, z_3, z_4)$. For example, z_1 has $\psi_1(\cdot; \theta_1) = \mathtt{AT}(\cdot; \mathtt{ball1}, \mathtt{roomb})$. ρ_1 retains the source location requirement, $\kappa_1 = 1$, and μ_1 uses *current-state* semantics. This represents a requirement verified from public location feedback, rather than a sequence of pickup, movement, and placement actions. The initial public observation reports all four balls and the robot in `rooma`, with both grippers free. Initial evidence changes all four atoms from unknown to unsatisfied, so $\mathcal{C}_0 = \{z_1, z_2, z_3, z_4\}$; this run selects $g_0^\star = z_1$. The prerequisites concerning the object's source, the robot's location, and a free gripper are already satisfied. The execution frontier therefore focuses on holding `ball1`. The table below shows observation excerpts and frontier changes over successive interactions. Judgments use the full public feedback, and all listed actions are generated by the base agent.

Feedback	Public observation excerpts	Goal frontier	Execution frontier or state focus

Initial state	`Ball1 is at rooma.`
	`Robby is at rooma.`
	`Right is free.`
		z_1	Holding `ball1`

Step 1: pick up	`Ball1 is carrying right.`	Retain z_1	Being at `roomb` while carrying `ball1`
Step 2: move	`Ball1 is carrying right.` `Robby is at roomb.`	Retain z_1	Prerequisites satisfied; focus on z_1 itself
Step 3: drop	`Ball1 is at roomb.`	Switch to z_2	Being at `rooma`, the source location of `ball2`

After step 1, the holding state changes from F to T, but $\sigma_1(z_1) = \mathsf{F}$, so the goal frontier remains at z_1. Backward traversal stops at the satisfied holding node. The only execution candidate is being at the destination while carrying the ball. The ball's departure from its source does not reactivate preparation for picking it up. After step 2, carrying the ball at the destination is also confirmed, and $\mathcal{R}_2 = \mathcal{F}_2 = \varnothing$. The goal z_1 is still unsatisfied, but its prerequisites hold. Thus, $\xi_2 = \langle z_1, \varnothing \rangle$, and guidance focuses on establishing and verifying the goal location relation. Only the placement feedback at step 3 makes $\sigma_3(z_1) = \mathsf{T}$. The goal candidates become $\{z_2, z_3, z_4\}$, and this run switches to z_2. The robot remains in `roomb`, while `ball2` is in `rooma`. The new execution frontier therefore focuses on reaching the object's source. It does not reuse the previous goal's satisfied prerequisite of carrying a ball at the destination. This example links $\mathcal{C}_t$, $\mathcal{F}_t$, and ξ_t to one recorded interaction sequence. Online judgments use only public interaction information, not the simulator's internal states used for offline evaluation.

C Experimental Setup and Evaluation Protocol

C.1 Agent Implementation and Paired Evaluation

Datasets and evaluation tasks. We follow the AgentBoard evaluation task lists (Ma et al., 2024), using 60 PDDL, 134 ALFWorld, and 90 ScienceWorld task instances. PDDL instances are identified by domain and problem index and cover Gripper, Blocksworld, Barman, and Tyreworld. ALFWorld uses the unseen-scene evaluation split `eval_out_of_distribution`, corresponding to `valid_unseen`, across six household task types. ScienceWorld uses the 16 task families and variations specified in the evaluation list. Task-family names and variation IDs identify its 90 instances. Table 9 summarizes the task composition.

Model deployment and inference. The main experiments use Qwen2.5-32B-Instruct, and the backbone sensitivity experiments use Qwen2.5-14B-Instruct. The 32B model runs on four NVIDIA RTX 3090 GPUs and the 14B model on two, each with 24 GB of memory. Both models are served through SGLang in `bfloat16` precision, with server seed 0 and `enable-deterministic-inference` enabled. Generation requests for both models use temperature 0, top-p 1, and client seed 0.

Table 9: Evaluation task composition. Counts apply to each model, agent paradigm, and experimental condition. The 16 ScienceWorld task families are grouped by content; parentheses give instance counts.

Dataset	Tasks	Composition
PDDL	60	Gripper (20), Blocksworld (10), Barman (20), and Tyreworld (10).
ALFWorld	134	Pick and place (24), heat and place (23), cool and place (21), clean and place (31), place two objects (17), and examine under a light (18).
ScienceWorld	90	Object identification (18), lifespan comparison (27), life-stage identification (5), chemical mixing (9), temperature and melting-point measurement (22), and phase changes (9).

Agent control flow. Act directly generates the next environment action from the goal, current observation, and interaction history. ReAct allows up to three explicit reasoning turns before an action. Our Plan-and-Execute implementation adapts the plan-then-solve structure of Wang et al. (2023a) to interactive environments. It first generates a high-level plan, then selects actions through an executor; our implementation adds replanning after consecutive rejections or prolonged lack of progress. Reflexion generates bounded textual reflections after repeated failures or stagnation and uses them in later decisions. The four paradigms share the environment interface, task goals, and dataset action instructions, while retaining their own planning, reasoning, and reflection calls. DualFront updates both frontiers and renders the current guidance before each action decision, without changing the base paradigm's action-generation logic.

Generation settings and paired protocol. Dataset-level prompt scaffolds are embedded in the paradigm-specific control templates listed in Appendix E. Generation limits are 256 tokens for an action, 512 for a high-level plan, and 192 for a reflection. Within each environment and paradigm, Baseline and DualFront use identical task orders, prompt scaffolds, control templates, action-format constraints, history renderers, context limits, decoding parameters, and termination criteria.

C.2 Evaluation Metrics and Statistical Aggregation

Task performance. For an evaluation set of N episodes, let $y_e \in \{0, 1\}$ indicate whether episode e succeeds within the action budget, and let $r_e \in [0, 1]$ be its highest recorded task-completion fraction. Success rate and progress rate are

$$\mathrm{SR} = \frac{1}{N}\sum_{e=1}^{N} y_e, \qquad \mathrm{PR} = \frac{1}{N}\sum_{e=1}^{N} r_e. \tag{11}$$

Following AgentBoard (Ma et al., 2024), PDDL progress is the fraction of goal facts that hold simultaneously. ALFWorld and ScienceWorld progress is the fraction of evaluation conditions confirmed as complete. r_e is the maximum over the interaction. Evaluation annotations are not provided as online input.

Execution costs and mechanism metrics. Let T_e be the number of environment actions in episode e; the mean is $\frac{1}{N}\sum_e T_e$. We also report mean LLM calls and input/output tokens per episode, together with total environment-rejected actions. LLM call counts include action generation and all internal requests made by each paradigm. Token counts come from server records; k denotes 1,000 tokens in the tables. Definitions and counting rules for RR, EMR, and ASRR appear in Appendix A.1.

Aggregation. In Table 1, Macro Avg. weights the three environments equally, and Mean weights the four agent paradigms equally. Let $\mathcal{M}$ and $\mathcal{D}$ denote the sets of paradigms and environments, respectively, and let Q be SR or PR expressed as a percentage. The mean gains are

$$\Delta Q_d = \frac{1}{|\mathcal{M}|}\sum_{m\in\mathcal{M}} \left(Q_{d,m}^{\mathrm{DualFront}} - Q_{d,m}^{\mathrm{Baseline}}\right), \qquad \Delta Q_{\mathrm{macro}} = \frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}} \Delta Q_d. \tag{12}$$

SR and PR gains are reported in percentage points. The relative reduction in a cost X is $(X_{\text{Baseline}} - X_{\text{DualFront}})/X_{\text{Baseline}}$. Pooled means in the cost tables are computed over all episodes. Differences and relative reductions are computed from unrounded data and then rounded to two decimal places; displayed means are rounded separately.

D Supplementary Experiments and Analysis

D.1 Token Overhead of Guidance

The code limits guidance length through `max_hint_chars`, with a default cap of 640 characters rather than a fixed token count. The fixed prefix `Current dynamic subgoal:` and adjacent line breaks encode to six tokens on their own; the dynamic guidance content is counted separately. We extract the added guidance blocks from actual LLM requests in the 12 DualFront main-experiment runs across three environments and four paradigms. Each block is encoded separately with the Qwen2.5-32B-Instruct tokenizer, including the prefix and adjacent line breaks but excluding special tokens. Of 22,608 model requests, 22,281 contain guidance. Each request containing guidance is one sample; these blocks total 580,757 tokens, with a mean of 26.07 and a range of 17–59 tokens. This measures the length of the added text block and excludes requests without guidance from the mean.

D.2 Effect of Guidance-Injection Frequency

Experimental setup and schedules. We use Qwen2.5-32B-Instruct on the same 60 PDDL tasks under each of the four agent paradigms, with a 30-step action budget. Both frontiers still update at every step; only the timing of guidance delivery changes. Combining a change trigger (on or off) with periods $K \in \{\infty, 4, 2\}$ gives the six schedules in Table 10(a). $K = \infty$ disables periodic injection, and every schedule delivers the first renderable guidance. When change-triggered injection is enabled, guidance is delivered whenever its current version differs from the last version delivered to the agent. A version change can result from a change in the selected goal, execution state, or rendered content; it is not limited to goal switches. For finite K, periodic injection occurs at environment-decision indices $t = K, 2K, \ldots$, anchored at $t = 0$. A change trigger does not reset this clock, and coincident triggers produce a single injection. The period counts environment interaction steps, not internal LLM calls. No guidance is injected when it cannot be rendered or task completion has been confirmed.

Results. Among the six schedules, C+P2 has the highest mean SR and PR, at 32.50% and 55.61%. These exceed start-only injection (S) by 5.83 and 5.01 percentage points, respectively. At both finite periods, combining change-triggered and periodic injection gives higher mean SR and PR than periodic injection alone: C+P4 exceeds P4, and C+P2 exceeds P2. These point estimates favor the combined schedules on this task batch. The response to injection schedules varies across paradigms. Switching from C to C+P2 raises Reflexion SR from 20.00% to 33.33%, a gain of 13.33 percentage points, but lowers P&E SR from 35.00% to 31.67%, a drop of 3.33 percentage points. For P&E, C gives the highest PR among the six schedules (57.22%); for Act and Reflexion, C+P2 gives the highest SR and PR. More frequent injection therefore does not improve every paradigm, and aggregate results do not replace paired comparisons within each paradigm.

D.3 Backbone Model: Qwen2.5-14B-Instruct

Experimental setup. We evaluate Baseline and DualFront with Qwen2.5-14B-Instruct (Qwen Team, 2025) under Act, ReAct, Plan-and-Execute, and Reflexion. We use the same 60 PDDL, 134 ALFWorld, and 90 ScienceWorld tasks as in the 32B experiments, giving 24 settings and 2,272 episodes. Runs use a 30-step environment-action budget, temperature 0, top-p 1, and seed 0. Tables 11 and 12 use the performance and cost metrics from the main text, with aggregation as defined in Appendix C.2.

Task performance. DualFront raises macro-averaged SR from 29.95% to 39.25% and PR from 51.06% to 59.17%, gains of 9.30 and 8.11 percentage points. PR improves in all 12 environment–paradigm pairs, and SR improves in 11. The exception is ALFWorld–Act, where SR decreases from

Table 10: Sensitivity to guidance-injection frequency. (a) Six schedules combine three periods with change-triggered injection enabled or disabled. (b) SR and PR (%); Mean averages over paradigms. Bold marks column maxima using unrounded values and indicates descriptive rankings only. Baseline uses no guidance; DualFront injects guidance at every step. Both reuse main-experiment results as references and are excluded from paired tests of the six schedules.

(a) Injection schedules

Period K	Change trigger: off	Change trigger: on
∞	S: Start-Only	C: Change-Only
4	P4: Periodic-4	C+P4: Change+Periodic-4
2	P2: Periodic-2	C+P2: Change+Periodic-2

(b) Performance across agent paradigms

Schedule	Act		ReAct		P&E		Reflexion		Mean	
	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑
Baseline	20.00	44.54	28.33	55.72	26.67	53.97	18.33	47.01	23.33	50.31
S	25.00	46.38	26.67	51.53	28.33	55.78	26.67	48.75	26.67	50.61
C	26.67	48.16	35.00	57.09	**35.00**	**57.22**	20.00	47.38	29.17	52.46
P4	26.67	48.85	31.67	54.62	28.33	52.47	21.67	48.88	27.08	51.20
C+P4	28.33	50.43	33.33	56.00	**35.00**	54.82	28.33	54.60	31.25	53.96
P2	23.33	49.48	35.00	56.55	28.33	54.16	28.33	52.30	28.75	53.12
C+P2	31.67	52.75	33.33	58.25	31.67	55.46	33.33	55.99	32.50	55.61
DualFront	**35.00**	**56.82**	**38.33**	**63.02**	33.33	56.80	**35.00**	**59.00**	**35.42**	**58.91**

54.48% to 51.49% while PR increases from 61.69% to 63.56%. These results extend the evaluation to a smaller backbone and show that gains still vary across environments and paradigms.

Table 11: Task performance with Qwen2.5-14B-Instruct. SR and PR are percentages; Macro Avg. weights environments equally, and Mean weights paradigms equally. Bold marks the better result in each pair; underlining marks the best of the eight configurations in each column. Shaded rows use DualFront.

Agent	PDDL		ALFWorld		ScienceWorld		Macro Avg.	
	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑	SR(%)↑	PR(%)↑
Act	21.67	42.14	**54.48**	61.69	16.67	43.97	30.94	49.27
+ DualFront	**35.00**	**52.24**	51.49	**63.56**	**26.67**	**54.90**	**37.72**	**56.90**
ReAct	23.33	46.00	63.43	74.94	3.33	31.66	30.03	50.87
+ DualFront	**31.67**	**51.84**	**73.13**	**82.52**	**12.22**	**50.52**	**39.01**	**61.63**
P&E	16.67	44.49	70.90	76.68	10.00	45.76	32.52	55.64
+ DualFront	**33.33**	**50.76**	**72.39**	**77.24**	**23.33**	**54.06**	**43.02**	**60.69**
Reflexion	16.67	41.53	54.48	62.31	7.78	41.48	26.31	48.44
+ DualFront	**31.67**	**51.37**	**58.96**	**67.16**	**21.11**	**53.86**	**37.24**	**57.46**
Mean: Baseline	19.58	43.54	60.82	68.91	9.44	40.72	29.95	51.06
Mean: DualFront	**32.92**	**51.55**	**63.99**	**72.62**	**20.83**	**53.33**	**39.25**	**59.17**
Δ (pp)	+13.33	+8.01	+3.17	+3.72	+11.39	+12.62	+9.30	+8.11

Execution cost and action reliability. Pooling all episodes, mean LLM calls decrease from 24.53 to 23.01, mean environment actions from 23.13 to 21.77, and RR from 35.83% to 29.43%. Mean input and output tokens also decrease (Table 12). These aggregate improvements do not hold for every setting. For ALFWorld–Act, mean calls increase from 19.14 to 19.41 and RR from 31.58% to 35.45%. Thus, the 14B results support aggregate improvements in task performance and execution efficiency, but individual metrics do not necessarily improve together within a setting.

Table 12: Execution cost and action reliability with Qwen2.5-14B-Instruct. Calls, actions, and tokens are episode averages. k denotes 1,000 tokens. Rejects are total counts, and RR divides rejects by all actions. All pools paradigms within an environment; Pooled aggregates all episodes. Bold marks the lower unrounded value in each pair; shaded rows use DualFront.

Env.	Agent	Setting	Mean count/episode		Mean tokens/episode		Action reliability	
			Calls↓	Actions↓	Input (k)↓	Output (k)↓	Rejects↓	RR(%)↓
PDDL	Act	Baseline	27.03	27.03	76.32	**0.288**	376	23.18
		+ DualFront	**25.20**	**25.20**	**74.29**	0.288	**321**	**21.23**
	ReAct	Baseline	31.38	27.53	**74.32**	1.286	361	21.85
		+ DualFront	**31.10**	**26.25**	75.37	**1.249**	**311**	**19.75**
	P&E	Baseline	28.80	27.80	74.19	0.520	272	16.31
		+ DualFront	**27.32**	**26.32**	**71.17**	**0.502**	**225**	**14.25**
	Reflexion	Baseline	27.17	27.13	63.16	0.284	364	22.36
		+ DualFront	**25.73**	**25.65**	**61.87**	**0.273**	**295**	**19.17**
	All	Baseline	28.60	27.38	72.00	0.595	1,373	20.90
		+ DualFront	**27.34**	**25.85**	**70.67**	**0.578**	**1,152**	**18.57**
ALFWorld	Act	Baseline	**19.14**	**19.14**	**23.53**	**0.177**	**810**	**31.58**
		+ DualFront	19.41	19.41	24.98	0.188	922	35.45
	ReAct	Baseline	20.32	17.84	22.26	0.750	520	21.76
		+ DualFront	**18.01**	**16.31**	**20.26**	**0.631**	**416**	**19.04**
	P&E	Baseline	15.82	14.74	**17.72**	**0.176**	**162**	**8.20**
		+ DualFront	**15.44**	**14.27**	17.90	0.178	**162**	8.47
	Reflexion	Baseline	19.65	18.87	21.51	0.185	494	19.53
		+ DualFront	**18.33**	**17.69**	**20.62**	**0.180**	**395**	**16.67**
	All	Baseline	18.73	17.65	21.26	0.322	1,986	21.00
		+ DualFront	**17.80**	**16.92**	**20.94**	**0.294**	**1,895**	**20.90**
ScienceWorld	Act	Baseline	26.79	26.79	71.95	0.223	1,491	61.84
		+ DualFront	**25.32**	**25.32**	**68.43**	**0.212**	**1,019**	**44.71**
	ReAct	Baseline	29.93	29.39	74.13	0.238	1,736	65.63
		+ DualFront	**27.73**	**27.50**	**70.16**	**0.216**	**1,101**	**44.48**
	P&E	Baseline	33.62	28.77	85.95	0.514	1,328	51.29
		+ DualFront	**30.23**	**26.22**	**78.26**	**0.439**	**1,083**	**45.89**
	Reflexion	Baseline	31.40	28.86	75.65	0.281	1,499	57.72
		+ DualFront	**28.31**	**26.06**	**68.67**	**0.260**	**1,029**	**43.88**
	All	Baseline	30.44	28.45	76.92	0.314	6,054	59.11
		+ DualFront	**27.90**	**26.27**	**71.38**	**0.282**	**4,232**	**44.74**
Pooled		Baseline	24.53	23.13	49.62	0.377	9,413	35.83
		+ DualFront	**23.01**	**21.77**	**47.43**	**0.350**	**7,279**	**29.43**

E PROMPT TEMPLATES

This appendix lists the prompt templates used by the four agent paradigms. Within each environment and paradigm, Baseline and DualFront share the control templates and dataset prompt content. Double braces denote fields filled at runtime. Dataset instructions, demonstrations, and guidance for actions, planning, and reflection are shown as placeholders. Each template is submitted as a user message; the system message comes from the dataset configuration.

Shared fields and guidance integration. `instruction` and `examples` contain task instructions and demonstrations. `history` and `observation` contain the interaction history and current public observation. `goal` contains the original task goal. When valid guidance is available, DualFront expands this field as shown in Prompt 1, where `hint` is the current guidance text. Initial planning uses the original goal. Later replanning and reflection calls append guidance only if the existing guidance still matches the current frontier.

Prompt 1: Shared Goal Field with DualFront Guidance

```
{{original_goal}}

Current dynamic subgoal:
{{hint}}
```

Prompt 2: Act

```
{{instruction}}

Official benchmark demonstration:
{{examples}}

Choose exactly one next environment action. Use the latest public
observation as the source of truth, ground every object and argument
in the goal or observations, and do not output a plan or private
chain of thought.

Dataset action and grounding guidance:
{{action_guidance}}

Task goal:
{{goal}}

Previous interaction:
{{history}}

Current observation:
{{observation}}

Your response must literally begin with `Action:`. Do not return only
the action body.
Respond with exactly one line:
Action: <one executable action>
```

`scratchpad` stores the reasoning generated within the current environment step. `response_instruction` specifies the output format, depending on whether the reasoning-turn limit has been reached.

Prompt 3: ReAct

```
{{instruction}}

Official benchmark demonstration:
{{examples}}

Use the ReAct method to choose the next action from the goal,
interaction history, and latest public observation. You may think
for several turns, but each environment turn must contain exactly
one action.

Task goal:
{{goal}}

Previous interaction:
{{history}}

Current observation:
{{observation}}

Reasoning in the current ReAct turn:
{{scratchpad}}

Dataset action and grounding guidance:
{{action_guidance}}

{{response_instruction}}
```

`response_instruction` before the reasoning-turn limit:

```
Respond with exactly one line that must literally begin with `Think:`
or `Action:`: Think: <reasoning> or Action: <one English action>
```

`response_instruction` at the reasoning-turn limit:

```
Stop thinking. Your response must literally begin with `Action:` and
contain exactly one line: Action: <one English action>
```

`old_plan` is the existing plan before planning, and `plan` is the high-level plan used for execution. `reason` identifies the planning or reflection trigger, and `reflections` contains the retained reflection memory.

Prompt 4: Plan-and-Execute (Planner)

```
{{instruction}}

Official benchmark demonstration:
{{examples}}

You are the Planner in a Plan-and-Execute agent. Build a short
high-level plan from only the task goal, public observation and
interaction history. The environment is partially observable, so use
exploration when a required entity or precondition has not been
observed. Do not invent identifiers.

Dataset planning guidance:
{{planning_guidance}}

Task goal:
{{goal}}

Current observation:
{{observation}}

Previous interaction:
{{history}}

Previous plan:
{{old_plan}}

Planning reason: {{reason}}

Return only a JSON array of concise English plan steps.
```

Prompt 5: Plan-and-Execute (Executor)

```
{{instruction}}

Official benchmark demonstration:
{{examples}}

You are the Executor in a Plan-and-Execute agent. The plan is
high-level guidance; the latest public observation is the source of
truth. Execute exactly one grounded action and do not invent
objects, identifiers or unavailable preconditions.

Dataset action and grounding guidance:
{{action_guidance}}

Task goal:
{{goal}}

High-level plan:
{{plan}}

Previous interaction:
{{history}}

Current observation:
{{observation}}

Your response must literally begin with `Action:`. Do not return only
the action body.
Respond with exactly one line:
Action: <one executable action>
```

Prompt 6: Reflexion (Action)

```
{{instruction}}

Official benchmark demonstration:
{{examples}}

Choose exactly one next environment action. Use the latest public
observation as the source of truth. Apply a previous reflection only
when its premise still matches the current state, and do not output
a new reflection or private chain of thought in this call.

Dataset action and grounding guidance:
{{action_guidance}}

Task goal:
{{goal}}

Bounded reflection memory:
{{reflections}}

Previous interaction:
{{history}}

Current observation:
{{observation}}

Your response must literally begin with `Action:`. Do not return only
the action body.
Respond with exactly one line:
Action: <one executable action>
```

Prompt 7: Reflexion (Reflection)

```
{{instruction}}

You are the reflection component of an interactive agent. Diagnose
one concrete, publicly supported cause of the recent failure or loop
and state one short corrective rule for later action selection. Do
not propose hidden objects, future states, benchmark subgoals or a
full new plan.

Dataset reflection guidance:
{{reflection_guidance}}

Task goal:
{{goal}}

Reflection trigger: {{reason}}

Existing reflection memory:
{{reflections}}

Previous interaction:
{{history}}

Latest public observation:
{{observation}}

Respond with one compact line:
Reflection: <failure cause and corrective rule>
```