THE GOAL GETS IN THE WAY: SUBGOAL RELAY FOR LONG-HORIZON LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language model agents often decompose long-horizon tasks into subtasks and use the final goal to organize and guide execution. Our motivation experiments reveal a contrast: using the final goal to clarify a subtask improves local completion, yet providing the goal again with the same explanation reduces completion. Direct goal guidance may lead the executor to reconsider earlier planning decisions, causing unproductive actions and rework. We therefore propose Subgoal Relay (SR) to limit the final goal's direct influence on local decisions while retaining its guidance for the current subtask. We aggregate goal information using attention weights from the current subgoal to the final goal. We fuse this information into the subgoal value cache and bound the update magnitude relative to the original values, forming a relay for local execution. The model draws on this relay and the retained context for its decisions, while attention access constraints block direct access to the original final-goal field. We evaluate SR on ALFWorld, ScienceWorld, and WebShop across four agent frameworks and six model backbones, measuring task performance and end-to-end execution cost. In the main experiments with Qwen3-8B under a shared attention backend, SR increases ALFWorld success rates by up to 7.8 percentage points and mean scores on ScienceWorld and WebShop by up to 4.6 and 8.7 points, respectively, while reducing mean end-to-end execution time by 2.5–11.1%.

1 INTRODUCTION

Language model agents carry out user tasks by calling tools and interacting with their environments (Yao et al., 2023; Trivedi et al., 2024; Erdogan et al., 2025; Agashe et al., 2025a). In long-horizon tasks, an agent must turn a user's overall request into a sequence of interdependent actions. Earlier actions establish the conditions and provide the information needed for later steps. New information gathered during execution may require changes to subsequent actions. An error at one step can propagate through these dependencies and lead the agent away from the intended outcome. Reliable task completion therefore requires a coherent sequence of local actions and continual adjustment based on environmental feedback.

Prior work organizes this process through explicit plans and subtasks that connect the final goal to individual actions (Zhang et al., 2025; Zhen et al., 2026; Wu et al., 2026). Task decomposition methods break complex tasks into manageable subtasks. They determine the decomposition and execution order based on the agent's capabilities or dependencies between tool calls (Prasad et al., 2024; Kim et al., 2024; Li et al., 2025). Feedback-driven planning methods use execution outcomes and new observations to revise plans, adapting subsequent work to the current environment state (Wang et al., 2023; Sun et al., 2023; Agashe et al., 2025b; Liu et al., 2025). Hierarchical planning and execution methods assign high-level planning and local execution to separate modules. They organize execution through global plans, subtask queues, or high-level blueprints (Erdogan et al., 2025; Agashe et al., 2025a; Jin et al., 2026). These methods typically use the final goal as a common basis for defining, coordinating, and revising subtasks.

However, in our motivation experiments, hiding the final goal from local execution raises whole-task completion, even though the goal is intended to keep the task on track. Under this direct guidance, the executor may reconsider earlier high-level planning decisions while refining subtasks or selecting actions. Such interference may divert execution from the current subgoal and cause unproductive actions and rework, adding unnecessary interaction steps. In a matched local comparison, using the

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.

final goal to clarify the requirements of the current subtask improves subgoal completion. However, when these requirements are held fixed, continuing to provide the final goal to the executor lowers the subgoal completion rate.

We therefore propose Subgoal Relay (SR) to retain the final goal's guidance for the current subtask while limiting its direct influence on local decisions. We aim to reduce deviations from the current subgoal and unnecessary rework. After encoding the full context, we aggregate goal information using the attention weights from the subgoal to the final goal. We fuse this information into the subgoal value cache and bound the update magnitude relative to the original values, forming a relay for subsequent decisions. During subsequent subtask refinement and action selection, we use attention access constraints to block direct access to the original final-goal field. The model draws on the relay and the retained context for these decisions.

We evaluate SR on ALFWorld, ScienceWorld, and WebShop across four agent frameworks and six model backbones. SR improves mean task performance across the evaluated framework–backbone pairs. End-to-end cost analysis also shows lower mean execution time after including relay construction costs. Behavioral analysis and ablations further show that SR can retain the final goal's guidance while reducing its interference with local decisions.

Our contributions are as follows:

- We identify contrasting effects of final-goal guidance in our motivation experiments. Using the goal to clarify subtask requirements improves local completion, yet providing it again during execution with the same explanation lowers completion.
- We propose SR, which combines subgoal-conditioned goal information transfer with attention access constraints and requires no training of additional parameters. We analyze the paths through which explicit goal information transfer contributes to response computation, along with the relative update bound on subgoal values.
- We evaluate SR across agent frameworks and model backbones, showing improved performance on long-horizon tasks. End-to-end cost analysis further shows that the savings in execution outweigh relay construction costs, reducing mean execution time.

2 Related Work

Planning and execution in long-horizon tasks. Prior work uses task decomposition and hierarchical collaboration to organize complex requests into a sequence of executable steps (Wang et al., 2023; Agashe et al., 2025a). ADaPT recursively decomposes tasks as needed based on the executor's capabilities (Prasad et al., 2024). To coordinate planning and execution, Plan-and-Act separates high-level plan generation from action execution (Erdogan et al., 2025). STEP-HRL uses hierarchical reinforcement learning to coordinate subgoal generation and action execution (Zhen et al., 2026). To adapt to changes during execution, Agent S2 updates the remaining plan using new environment observations at the end of each subgoal's execution (Agashe et al., 2025b). ReCAP focuses on context continuity during recursive execution. It executes subtasks in a shared context and reinjects the parent plan when returning to the parent task, connecting subtask outcomes to subsequent planning (Zhang et al., 2025).

Context management in long-horizon tasks. As interaction histories grow, prior work organizes these histories and environment observations to reduce redundancy and retain information needed for subsequent actions (Zhou et al., 2026; Wan et al., 2026). For long-horizon web tasks, AgentFold summarizes individual interactions or sequences of exploration steps at different levels of detail, preserving key findings for subsequent actions (Ye et al., 2026). ACON compares execution outcomes under full and compressed contexts. It uses this comparison to refine the summarization rules for interaction histories and environment observations to reduce the loss of key information (Kang et al., 2026). ATA masks attention between mutually exclusive subtasks during encoding and dynamically suppresses attention to inactive subtasks during generation, reducing interference in compositional instructions (Lv et al., 2026). HiAgent groups the interaction history by subgoal to organize working memory. It replaces the interaction history of completed subgoals with summaries and retains a detailed trajectory for the current subgoal. It also supports retrieval of past trajectories as needed (Hu et al., 2025).

3 The Effect of the Final Goal on Local Execution

The final goal is often used to help the executor understand its current work within the overall task. To examine this guidance once the subgoal is specified, we compare local execution with the final goal provided (Full) or hidden (Hide). Both conditions retain the current subgoal, necessary state information, interaction history, and applicable constraints. Root-level planning and revisions based on upward feedback retain access to the final goal. Hiding the final goal instead raises whole-task completion from 45.0% to 48.3%, a gain of 3.3 percentage points (Figure 1(a)).

This improvement does not imply that the goal's information has no value. We further examine its role at matched subgoal execution entry points. The same model generates local explanations with or without the final goal, clarifying the current work's scope, completion conditions, and requirements. We provide each explanation alongside the current subgoal and necessary context. Both conditions share the starting state and execution budget, and neither receives the final goal during execution. We assess completion using predefined environment-state criteria. Using the final goal to generate the explanation raises subgoal completion from 56.7% to 63.3%, a gain of 6.7 percentage points (Figure 1(b)).

We then reuse the explanation generated with the final goal and test whether additionally providing the goal during execution affects completion. The starting state and all other inputs remain fixed. This additional input instead lowers subgoal completion from 63.3% to 58.3%, a drop of 5.0 percentage points (Figure 1(b)). To understand what changes in execution accompany this decline, we inspect the corresponding trajectories. When additionally given the final goal, the executor more often moves unnecessarily to work on other stages before completing the current subgoal. The proportion of local executions with this behavior rises from 10.0% to 15.0% (Figure 1(c)). Across these two comparisons, using the final goal to generate a local explanation improves completion, yet providing it again to the executor with the same explanation lowers completion.

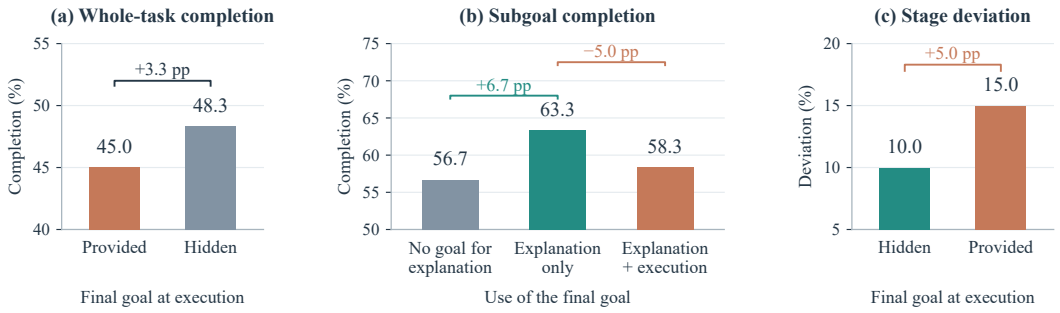

Figure 1: **Effects of final-goal use on execution.** (a) Whole-task completion with ReCAP/Qwen3-4B on 60 tasks sampled from the ALFWorld training split. (b) Subgoal completion when the final goal is unused, used only to generate a local explanation, or also provided during execution. All three conditions include an explanation. (c) Local executions with at least one unnecessary stage deviation. The last two conditions in (b) and both in (c) share the same goal-conditioned explanation. Each task contributes one local entry, evaluated once per condition. Vertical axes start above zero to show differences, so visible bar lengths do not represent rate ratios. Brackets show differences, not significance. pp denotes percentage points, and all differences use unrounded rates. Paired transitions and protocols are in Appendix C.

4 Method: Subgoal Relay

The final goal helps high-level planning determine the current subgoal. Once that subgoal is set, however, continued direct guidance from the final goal may interfere with the execution of the assigned subtask. We propose Subgoal Relay (SR) to retain the final goal's guidance for the current subtask while limiting its direct influence on local action selection. SR aggregates information using the current subgoal's attention to the final goal and transfers it into the subgoal representations with a bounded update. High-level planning and upward revisions based on feedback retain their original

access to the final goal. For calls that further refine an established subgoal or generate actions, we restrict the response's direct access to the original final-goal field while preserving access to the updated subgoal representations and interaction context.

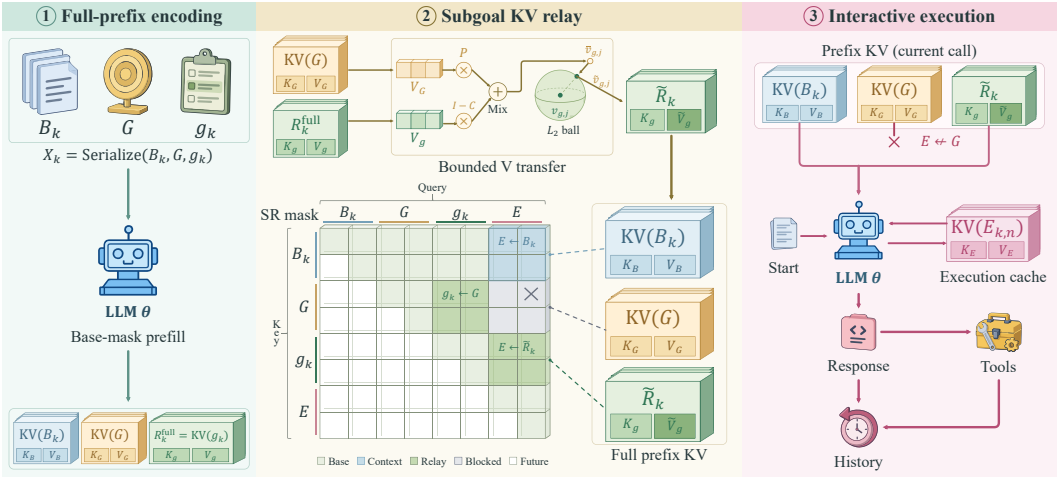

Figure 2: **Overview of Subgoal Relay (SR).** (1) **Full-prefix encoding.** We encode the context B_k, final goal G, and current subgoal g_k under the base mask. (2) **Subgoal KV relay.** Subgoal attention guides goal-value transfer into the subgoal cache, with updates bounded relative to the original values. Keys remain unchanged. The SR mask blocks direct response access to goal positions. (3) **Interactive execution.** The model generates responses using the relay and retained context. Responses and tool feedback update the history for the next call, when the prefix is re-encoded and the relay rebuilt.

4.1 Tasks and Interaction Context

We describe SR using a recursive task controller that organizes execution by progressively refining subtasks and revises plans using feedback returned to higher levels. Consider the k-th model call, in which the controller processes an established subgoal g_k in a downward call. Let G denote the final goal. The input contains G, g_k, observations, and interaction history. We use *response* to refer collectively to the reasoning text, subtask plans, or action instructions produced by the model. A subgoal may require multiple calls, and feedback from tool execution is included in subsequent calls.

We denote the tokenized input prefix and the response tokens already encoded in this call by

$$X_k = \text{Serialize}(B_k, G, g_k), \qquad \mathbf{x}_{k,n} = [X_k; E_{k,n}] = (x_1, \ldots, x_N). \tag{1}$$

Here, X_k is the input prefix. B_k contains the context outside the designated final-goal and current-subgoal fields, including observations, interaction history, tool descriptions, and applicable constraints. Serialize arranges the fields using the baseline method's message order and chat template, without assuming that the two goal fields are adjacent. $E_{k,n}$ contains the n response tokens already encoded in this call, including the response-start marker from the chat template. N is the current sequence length, and x_i is the i-th token.

Let I_B, I_G, I_g, and I_E denote the position sets for the context, designated final-goal field, current-subgoal field, and current response, respectively. The first three sets are disjoint and cover the input prefix. I_E grows as the response is generated. Appendix B.1 describes field localization and template boundaries.

4.2 Autoregressive Attention

Within a model call, the model first encodes the input prefix and then generates the response token by token. For an attention head in a given layer, let Q, K, and V be the query, key, and value matrices, and let M be the visibility mask. Scaled dot-product attention (Vaswani et al., 2017) computes the

attention weights and output as

$$A(M) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_h}} + M\right), \qquad O(M) = A(M)V. \tag{2}$$

Here, d_h is the query and key dimension. For query position i and key position j, $M_{ij} = 0$ permits access, whereas $M_{ij} = -\infty$ blocks the connection. The base mask M^{base} includes the causal constraint $j \le i$ and the model's original visibility rules.

During prefix encoding, field positions and the base mask jointly determine which final-goal positions the current subgoal can attend to. During response generation, the model accesses the key and value representations of the input prefix and tokens already generated in the current call through the key–value (KV) cache. Whether the response can directly attend to the final-goal field depends on the mask used at this stage.

4.3 The Subgoal Relay Mechanism

We construct the relay at the current subgoal's existing token positions, without adding tokens. After encoding the full prefix, we update the value cache at these positions and restrict direct access to the original final-goal field during response generation.

Representing goal information at subgoal positions. Encoding the full input prefix under the base mask M^{base} yields the KV representations at each layer and the attention weights from subgoal positions to the final goal. No transferred information is written during this encoding process. We denote the resulting layer-wise subgoal KV representations by

$$\mathcal{R}_k^{\mathrm{full}} = \left\{ \left(K_{I_g}^{\mathrm{full},(\ell)}, V_{I_g}^{\mathrm{full},(\ell)} \right) \right\}_{\ell=1}^{L}. \tag{3}$$

Here, L is the number of model layers. $K_{I_g}^{\mathrm{full},(\ell)}$ and $V_{I_g}^{\mathrm{full},(\ell)}$ are computed from the subgoal representations entering layer ℓ and include all KV heads in that layer.

At a given layer, the keys and values at a subgoal position are formed from the layer's input representation, which may already contain goal information from preceding layers. Attention in this layer then aggregates information from visible final-goal positions. This readout contributes to the position's output representation but does not change the value cache already formed in that layer.

Subgoal-conditioned goal transfer. We aggregate the final-goal information accessible to the current subgoal using its attention weights from the original prefix encoding.

Consider one attention head in a given layer, omitting call, layer, and head indices. For $j \in I_g$ and $r \in I_G$, let $\mathbf{v}_{g,j}$ be the original value vector at subgoal position j and $\mathbf{v}_{G,r}$ the value vector at final-goal position r. Define

$$P_{jr} = A(M^{\mathrm{base}})_{jr}, \qquad c_j = \sum_{r \in I_G} P_{jr}, \qquad \mathbf{t}_j = \sum_{r \in I_G} P_{jr} \mathbf{v}_{G,r}. \tag{4}$$

P_{jr} is the native attention weight from subgoal position j to final-goal position r. c_j is the attention mass that this query assigns to the final-goal field, and $\mathbf{t}_j$ is the resulting goal readout. We retain the native normalization over all visible positions rather than renormalizing within the final-goal field, avoiding amplification of an originally small attention mass on the goal.

To incorporate goal information into the existing subgoal representation, we use $\mathbf{v}_{g,j}$ as the reference for fusion. The goal readout $\mathbf{t}_j$ retains the native attention weights. We assign the remaining attention mass $1 - c_j$ from other visible positions to the original value, forming the candidate

$$\overline{\mathbf{v}}_{g,j} = (1 - c_j)\mathbf{v}_{g,j} + \mathbf{t}_j. \tag{5}$$

Here, $\mathbf{v}_{g,j}$ and $\mathbf{v}_{G,r}$ belong to the same value head in the same layer. The coefficients are nonnegative and sum to one, so the candidate is a convex combination of the original subgoal value and the visible final-goal values.

Bounding the transfer magnitude. The attention mass determines the total weight of goal values in the candidate, but does not by itself bound the distance from the original subgoal value. To control the update written to the cache, we use a shared relative budget $\rho \in [0, 1)$ and require each subgoal value to change by at most ρ times its original norm. Let $\mathbf{d}_j = \overline{\mathbf{v}}_{g,j} - \mathbf{v}_{g,j}$. When $\|\mathbf{d}_j\|_2 > 0$, the update is

$$\beta_j = \min\left(1, \frac{\rho\|\mathbf{v}_{g,j}\|_2}{\|\mathbf{d}_j\|_2}\right), \qquad \widetilde{\mathbf{v}}_{g,j} = \mathbf{v}_{g,j} + \beta_j \mathbf{d}_j. \tag{6}$$

When $\mathbf{d}_j = \mathbf{0}$, we set $\beta_j = 1$ and retain the original value. This update is equivalent to projecting the candidate onto a Euclidean ball centered at the original value. Candidates within the budget are retained. For candidates outside the budget, the displacement from the original value is scaled down so that

$$\|\widetilde{\mathbf{v}}_{g,j} - \mathbf{v}_{g,j}\|_2 \le \rho\|\mathbf{v}_{g,j}\|_2.$$

After bounding the update, we write the updated values to the corresponding subgoal positions while retaining the original keys used to match response queries. This gives the relay accessed during response generation:

$$\widetilde{\mathcal{R}}_k = \left\{\left(K^{\mathrm{full},(\ell)}_{I_g}, \widetilde{V}^{(\ell)}_{I_g}\right)\right\}_{\ell=1}^{L}. \tag{7}$$

All keys and the values at other prefix positions remain unchanged. Appendix B.3 gives the construction for shared KV heads.

Reading goal content through the relay. After updating the subgoal cache, we block attention connections from response positions I_E to original goal positions I_G, leaving all other visibility rules unchanged:

$$M^{\mathrm{SR}}_{ij} = \begin{cases} -\infty, & j \in I_G \ \wedge \ i \in I_E, \\ M^{\mathrm{base}}_{ij}, & \text{otherwise.} \end{cases} \tag{8}$$

This constraint applies to all self-attention layers and heads, starting from the first prediction in the current response. The model can still access the updated subgoal cache, the remaining context, and tokens already generated in the current response.

Explicit goal-value transfer connects two stages of attention. During prefix encoding, subgoal queries determine how goal values are aggregated. During response generation, response queries access the transferred content through the updated subgoal values. The subgoal cache stays fixed within a call, while response attention to these positions can change as tokens are generated.

To describe how these stages jointly determine the readout of transferred content, consider again one attention head in a given layer. Stack the original subgoal values and final-goal values by position into matrices V_g and V_G, denote the updated subgoal value matrix by $\widetilde{V}_g$, and write $P = [P_{jr}]$. Let S and C be diagonal matrices containing β_j and c_j for each subgoal position, respectively, and let I be the identity matrix of the same size.

Let $\mathbf{a}_{i,g}$ denote the attention weights from response position i to subgoal positions under the SR mask, taken from the attention distribution over all visible positions. The attention output contributed by subgoal positions expands as

$$\mathbf{o}_{i,g} = \mathbf{a}_{i,g}\widetilde{V}_g = \mathbf{a}_{i,g}(I - SC)V_g + \underbrace{\mathbf{a}_{i,g}SP}_{\text{coefficients via subgoals}} V_G. \tag{9}$$

The coefficients $\mathbf{a}_{i,g}SP$ combine the attention weights from both stages with the magnitude-control coefficients. Their sum is at most the attention mass that the response assigns to subgoal positions. Appendix A further explains the transfer magnitude and the path through which goal values are read.

Call scope and relay updates. In recursive task control, we apply SR to local calls that further refine an established subgoal or generate actions. Root-level planning and upward revisions based on feedback retain the baseline method's access to the final goal. For each call using SR, the model reconstructs the relay from that call's input prefix and holds it fixed throughout response generation. Appendices B.1 and B.2 specify call eligibility and the cache procedure.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate SR on ALFWorld, ScienceWorld, and WebShop, covering household tasks, scientific procedures, and online shopping (Shridhar et al., 2021; Wang et al., 2022; Yao et al., 2022). We use 134 unseen ALFWorld tasks, 211 ScienceWorld tasks, and 500 WebShop test instructions. The main experiments use Qwen3-8B with Plan-and-Execute, ReCAP, HiAgent, and TDP (Prasad et al., 2024; Zhang et al., 2025; Hu et al., 2025; Li et al., 2026). Plan-and-Execute is a controlled in-house implementation, and TDP is a reproduction from the paper. Each Base/+SR pair shares model weights, planning and control procedures, environment interfaces, and execution budgets. All parameters are frozen, and Qwen3's built-in thinking is disabled. SR parameters are selected on development data and fixed for testing.

ALFWorld reports success rate, while ScienceWorld and WebShop report their task scores. We include all attempts and report means and sample standard deviations over five paired seeds. End-to-end time includes model computation, SR construction, online processing, and environment interaction. Additional backbones, configurations, and timing details appear in Appendices D.1 and C.1.

5.2 TASK PERFORMANCE ACROSS AGENT FRAMEWORKS

Table 1: Paired Base/+SR results with Qwen3-8B. ALFWorld reports success (%), ScienceWorld mean final score, and WebShop mean purchase reward $\times 100$. T is mean end-to-end time per task (seconds), including SR computation and environment interaction. Values are means $\pm$ sample standard deviations over five paired evaluations.

		ALFWorld		ScienceWorld		WebShop	
Method	Variant	Success (%) ↑	T (s) ↓	Score ↑	T (s) ↓	Score ↑	T (s) ↓
Plan-and-Execute	Base	62.4 ± 2.0	44.7 ± 1.4	22.1 ± 2.6	80.5 ± 2.5	38.8 ± 1.7	22.9 ± 1.0
	+SR	69.0 ± 1.7	40.6 ± 1.1	25.9 ± 2.3	75.7 ± 2.3	45.2 ± 1.5	21.0 ± 0.9
ReCAP	Base	75.8 ± 1.8	53.9 ± 1.6	28.4 ± 2.8	107.3 ± 3.8	47.1 ± 1.6	30.5 ± 1.4
	+SR	83.6 ± 1.5	47.9 ± 1.4	33.0 ± 2.4	100.2 ± 3.5	55.8 ± 1.4	27.8 ± 1.3
HiAgent	Base	60.9 ± 2.1	48.9 ± 1.0	24.9 ± 2.5	88.2 ± 3.1	42.3 ± 1.9	26.8 ± 1.2
	+SR	66.6 ± 2.3	45.9 ± 1.2	28.8 ± 2.7	86.0 ± 2.9	47.7 ± 1.7	25.9 ± 1.2
TDP	Base	71.5 ± 1.7	64.3 ± 1.8	32.1 ± 2.2	94.9 ± 3.2	44.2 ± 1.8	31.5 ± 1.7
	+SR	77.0 ± 1.7	60.1 ± 1.5	36.3 ± 2.5	86.8 ± 3.0	51.5 ± 2.0	28.5 ± 1.5

SR improves task performance for all four frameworks in every environment (Table 1). ALFWorld success increases by 5.5–7.8 percentage points, while ScienceWorld and WebShop scores increase by 3.8–4.6 and 5.4–8.7 points. Gains extend to ReCAP and HiAgent, which already organize execution through recursive planning and subgoal-based memory. The five additional backbones show the same direction of mean task gains (Appendix D.1).

Under the shared attention backend, mean end-to-end time decreases by 2.5–11.1%, including transfer overhead. Paired task-metric intervals are above zero in all framework–environment pairs. Time gains are less uniform: the HiAgent/ScienceWorld interval crosses zero (Appendix D.11).

5.3 COMPONENT ABLATIONS AND GOAL HANDOFF

Group A shares the full-prefix encoding and isolates how goal information reaches local decisions. Full leaves the cache and access unchanged. Mask-Only blocks direct goal access without value transfer. Transfer-Only updates subgoal values while leaving direct access open. SR transfers goal information through the subgoal and blocks the direct route.

SR has higher mean task performance than either single component in all three environments. Its task-gain interval over Mask-Only is above zero on WebShop and crosses zero on ALFWorld and ScienceWorld (Appendix D.10). Under the per-row norm-matching control, subgoal-conditioned weights

Table 2: ReCAP/Qwen3-8B ablations and handoff controls. Metrics follow Table 1. Group A retains the original input construction. Group B changes the goal handoff. Means over five paired evaluations are shown. Full is Base, and Full/SR values are reused from the main table. All online overhead is included in T.

	ALFWorld		ScienceWorld		WebShop	
Condition	Success (%) ↑	T (s) ↓	Score ↑	T (s) ↓	Score ↑	T (s) ↓
A. Component ablations: original input construction						
Full	75.8	53.9	28.4	107.3	47.1	30.5
Mask-Only	79.6	50.4	30.1	103.6	51.6	29.1
Transfer-Only	77.6	55.4	29.3	113.6	49.1	31.9
SR	83.6	47.9	33.0	100.2	55.8	27.8
B. Goal handoff strategies: modified handoff inputs						
Hide	77.2	53.4	29.2	107.7	48.4	30.8
TH	80.6	64.7	31.4	134.8	53.2	36.6

yield higher mean local completion than uniform weights (73.9% versus 68.4%, Appendix D.7). We next examine what guidance transfer preserves when direct goal access is blocked.

Hide removes the goal before encoding. Textual Handoff (TH) first generates a local instruction from the goal and subgoal, then hides the goal during execution. TH improves on Hide. SR outperforms TH in mean task scores and time. Paired intervals support SR's time advantage in all three environments, but task-gain intervals cross zero on ALFWorld and ScienceWorld. Further control strategies appear in Appendix D.4.

5.4 Decision Boundaries and Necessary Guidance

We test whether SR limits interference while retaining necessary guidance. The boundary group tests adherence to the current subgoal, using 384 ScienceWorld entries from 96 tasks. The necessary-guidance group tests requirements stated only in the final goal, using 256 entries from 64 tasks. Conditions share entry states and contexts, with five repetitions per entry.

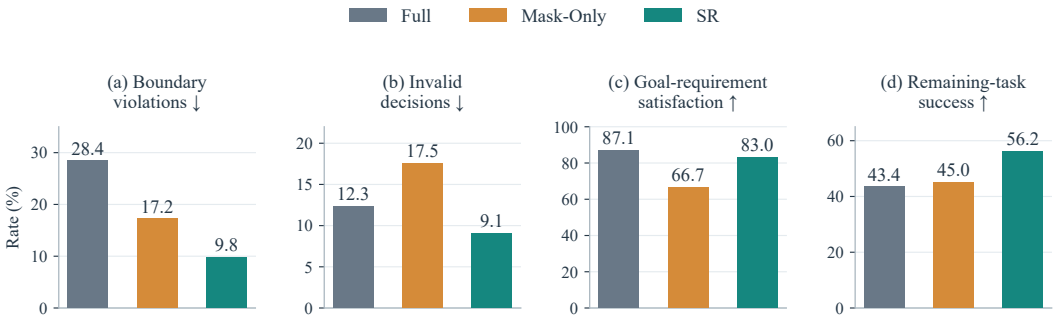

Figure 3: ReCAP/Qwen3-8B diagnostics on ScienceWorld. (a,b) First-call boundary violations and invalid decisions. (c) Goal-requirement satisfaction during local execution, including valid progress or justified return of control. (d) Subsequent execution completes the full task. Arrows indicate improvement directions. The two groups use separate entries. Rates average repetitions and entries within tasks, then weight tasks equally. Intervals are omitted. Definitions are in Appendix E.1.

Mask-Only reduces boundary violations but increases invalid decisions (Figure 3(a,b)). It also lowers goal-requirement satisfaction from Full's 87.1% to 66.7% (c), exposing the loss of necessary guidance when direct access is blocked. SR recovers satisfaction to 83.0% while reducing boundary violations and invalid decisions to 9.8% and 9.1%. This partial recovery accompanies higher remaining-task success: 56.2%, versus Full's 43.4% and Mask-Only's 45.0% (d). The joint pattern supports the role of the relay: retaining necessary guidance while limiting interference, so local constraints can support subsequent task completion. Definitions and annotation details appear in Appendix E.3.

5.5 Task Length and Execution Cost

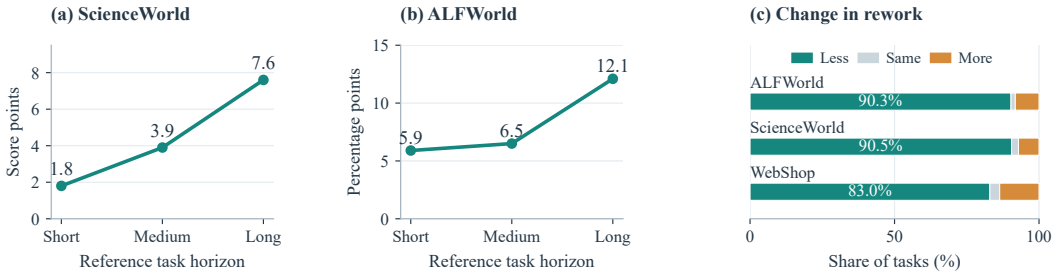

Figure 4: Task length and rework for ReCAP/Qwen3-8B. (a,b) SR−Full gains by independent reference action count, with separate axes for score points and success-rate percentage points. (c) Shares of tasks with fewer, equal, or more rework actions under SR than Full. Labels give the less-rework share. Exact counts and grouping details are in Appendix D.13. Uncertainty intervals are not shown.

SR's gains are larger in the longer reference-horizon groups, and rework decreases on most tasks (Figure 4). Groups use independent reference action counts, rather than trajectories produced by SR. Rework counts actions that correct deviations or repeat work after failed acceptance checks. It excludes normal exploration and planned stage transitions.

Table 3: Execution workload and model time for ReCAP/Qwen3-8B. Per-task means are Full / SR, with overhead reported only for SR. Calls include all online roles. Output/call instead reports total output tokens divided by total calls. Model time includes relay construction once and excludes tool waiting. All attempts are included.

Environment	Model calls	Output/call (tokens)	Rework actions	SR overhead (s)	Model time (s)
ALFWorld	29.0 / 25.5	67.5 / 67.4	2.6 / 1.5	0.9	41.3 / 36.4
ScienceWorld	89.1 / 82.3	45.8 / 45.5	7.8 / 3.7	3.1	85.5 / 79.0
WebShop	16.6 / 15.1	53.1 / 53.2	1.8 / 0.9	0.6	18.6 / 16.5

Table 3 links execution cost to fewer model calls with nearly unchanged output length per call. On ScienceWorld, calls decrease from 89.1 to 82.3 and model time from 85.5 to 79.0 seconds, including 3.1 seconds of transfer. Savings reflect less repeated work and accumulated context processing. Full and SR use the same attention backend. Appendix D.12 examines how faster baseline computation affects net savings. Grouping details and cost breakdowns appear in Appendices D.13 and D.8.

Truncating recorded trajectories at matched action or time cutoffs also shows greater task progress with SR. At 30 seconds on WebShop, SR scores 32.5, compared with Full's 24.5 and TH's 21.1. Instruction generation leaves TH less time for execution. Complete curves appear in Appendices D.14 and D.6.

6 Conclusion

We find that the final goal can both help and hinder execution in long-horizon tasks, depending on how it is used in decision-making. Based on this finding, we propose Subgoal Relay (SR), which passes the final goal's guidance through the current subgoal while limiting its direct influence on local decisions. Evaluations across agent frameworks and model backbones show that SR improves task performance. Execution cost analysis further shows that mean end-to-end execution time decreases even after including relay construction costs. These results show that information access constraints within the model can preserve the final goal's guidance while reducing its interference with local decisions.

AI USE STATEMENT

We used generative AI tools to assist with language polishing, code development, and LaTeX formatting corrections. The authors reviewed the outputs and take full responsibility for the final content.

REPRODUCIBILITY STATEMENT

Appendix A explains the relay mechanism and its relative update bound. Appendix B describes input-field localization, cache updates, and attention implementation. Appendix C describes the experimental protocols and timing procedures. Appendix D reports supplementary results and statistical analyses. Appendix E defines the behavior diagnostics and describes label review and annotation agreement.

REFERENCES

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

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S2: A compositional generalist-specialist framework for computer use agents. In *COLM*, 2025b.

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 *ICML*, volume 267 of *PMLR*, pp. 15419–15462, 2025.

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 *ACL*, pp. 32779–32798, 2025.

Hongbo Jin, Rongpeng Zhu, Jiayu Ding, Guibo Luo, and Ge Li. HiMAC: Hierarchical macro-micro learning for long-horizon LLM agents. *arXiv preprint arXiv:2603.00977v2*, 2026.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin Inan, Lukas Wutschitz, Yanzhi Chen, Robert A. Sim, and Saravanakumar Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/66270.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *ICML*, volume 235 of *PMLR*, pp. 24370–24391, 2024.

Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning in multi-agent systems. In *ICLR*, pp. 19495–19517, 2025.

Yunfan Li, Bingbing Xu, Xueyun Tian, Xiucheng Xu, and Huawei Shen. Beyond entangled planning: Task-decoupled planning for long-horizon agents. *arXiv preprint arXiv:2601.07577*, 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 *ACL*, pp. 20330–20354, 2025.

Fangrui Lv, Yulei Qin, Ruixin Hong, Jian Liang, Jinyang Wu, Ke Li, Xing Sun, and Changshui Zhang. Attend to the active: Structure-aware dynamic attention in LLMs for compositional instruction following. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/592ca58abdc87bd0ac92e9e83192d303-Abstract-Conference.html.

Art B. Owen. The pigeonhole bootstrap. *The Annals of Applied Statistics*, 1(2):386–411, 2007.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models. In *Findings of NAACL*, pp. 4226–4252, 2024.

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

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with language models. In *NeurIPS*, volume 36, pp. 58202–58245, 2023.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *ACL*, pp. 16022–16076, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, volume 30, pp. 5998–6008, 2017.

Guangya Wan, Mingyang Ling, Xiaoqi Ren, Rujun Han, Sheng Li, and Zizhao Zhang. COMPASS: Enhancing agent long-horizon reasoning with evolving context. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 3360–3380. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.152. URL https://doi.org/10.18653/v1/2026.acl-long.152.

Ruoyao Wang, Peter A. Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *EMNLP*, pp. 11279–11298, 2022.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with LLMs enables open-world multi-task agents. In *NeurIPS*, volume 36, pp. 34153–34189, 2023.

Zhe Wu, Donglin Mo, Hongjin Lu, Junliang Xing, Jianheng Liu, Yuheng Jing, Kai Li, Kun Shao, Jianye Hao, and Yuanchun Shi. K^2-Agent: Co-evolving know-what and know-how for hierarchical mobile device control. In *ICLR*, pp. 108447–108480, 2026.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*, volume 35, pp. 20744–20757, 2022.

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 *ICLR*, 2023.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Wenbiao Yin, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Jingren Zhou, Siheng Chen, and Yong Jiang. AgentFold: Long-horizon web agents with proactive context folding. In *International Conference on Learning Representations*, pp. 154754–154776, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/fa948624dfde013671e72c1a7ca4aebc-Abstract-Conference.html.

Qingru Zhang, Chandan Singh, Liyuan Liu, Xiaodong Liu, Bin Yu, Jianfeng Gao, and Tuo Zhao. Tell your model where to attend: Post-hoc attention steering for LLMs. In *ICLR*, pp. 42411–42430, 2024.

Zhenyu Zhang, Tianyi Chen, Weiran Xu, Alex Pentland, and Jiaxin Pei. ReCAP: Recursive context-aware reasoning and planning for large language model agents. In *NeurIPS*, volume 38, pp. 95158–95186, 2025.

Shuai Zhen, Yanhua Yu, Ruopei Guo, Nan Cheng, and Yang Deng. Hierarchical reinforcement learning with augmented step-level transitions for LLM agents. In *ACL*, pp. 7035–7053, 2026.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, pp. 58413–58438, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5fc8b3bdfbb9167b5144df5d3fae4616-Abstract-Conference.html.

A Supplementary Explanation of the Relay Mechanism

This appendix explains how explicit transfer differs from native prefix encoding, how its magnitude is controlled, and how responses read the transferred content.

A.1 Native Encoding and Explicit Transfer

Native prefix encoding already incorporates visible context into subgoal representations. Within a given layer, however, the goal information aggregated by attention updates the output representation, whereas the value cache has already been formed from the input representation. Explicit transfer incorporates this goal readout into the subgoal value cache of the same layer for subsequent response generation.

Each layer's transfer quantities come from the same unmodified full-prefix computation. After encoding the entire prefix, the model updates subgoal values while preserving all keys and other prefix values. Updates do not feed back into prefix computation, so native context encoding precedes the cache update for response generation.

A.2 Controlling the Transfer Magnitude

Goal attention mass determines the weight of goal values in the fusion candidate, while the relative update budget limits its deviation from the original subgoal value. Even a small attention mass can accompany a large distance between goal values and the original value, so native attention weights alone do not bound the actual update.

Equation (6) updates the original value in the direction of the fusion candidate. It retains candidates within the budget and scales down the displacement for those outside it. Expanding the update gives

$$\widetilde{\mathbf{v}}_{g,j} = (1 - \beta_j c_j)\mathbf{v}_{g,j} + \beta_j \sum_{r \in I_G} P_{jr}\mathbf{v}_{G,r}. \tag{A.1}$$

Since $P_{jr} \geq 0$, $c_j = \sum_r P_{jr} \in [0, 1]$, and $\beta_j \in [0, 1]$, all coefficients are nonnegative and sum to one. When a candidate exceeds the budget, reducing β_j lowers the weight on the goal-value terms and increases the weight on the original value. This keeps the update closer to the original subgoal value while preserving the convex combination.

In exact arithmetic, this update satisfies the relative update bound in the main text. Appendix B.3 describes how writes at finite precision are handled. When $\rho = 0$, subgoal values remain unchanged, and the method reduces to response-side goal masking after full-prefix encoding.

A.3 Reading Transferred Content through Subgoals

Explicit transfer connects prefix encoding and response generation. During prefix encoding, subgoal queries aggregate goal values through P, and the magnitude coefficients in S control how much of these values is incorporated into the subgoal cache. During response generation, new queries read the updated subgoal values through attention weights $\mathbf{a}_{i,g}$. The explicit goal-value term in Eq. (9) thus has coefficients $\mathbf{a}_{i,g}SP$.

Let a_{ij} be the component of $\mathbf{a}_{i,g}$ at subgoal position j, and let $\mathbf{1}$ be an all-ones column vector whose length equals the number of goal positions. The coefficient sum satisfies

$$\mathbf{a}_{i,g}SP\mathbf{1} = \sum_{j \in I_g} a_{ij}\beta_j c_j \leq \sum_{j \in I_g} a_{ij}. \tag{A.2}$$

The added transfer term therefore enters the attention output through the response's attention to subgoal positions. Its coefficient sum does not exceed the attention mass assigned to these positions. Within a call, the updated subgoal cache remains fixed, while the response's attention weights vary with the query.

This bound applies to the explicit transfer term. Goal information already encoded in the original subgoal representations and other context can still contribute to the response through their respective visible positions.

B Implementation and Reproducibility Details

This appendix specifies SR's call scope, input-field localization, and cache update procedure, along with the handling of shared KV heads and finite-precision computation.

B.1 Call Scope and Input-Field Localization

For the ReCAP interface, SR applies to downward calls at non-root nodes whose current subgoal differs from the root task. These calls further refine an established subgoal or generate actions. Root-level planning and upward revisions based on feedback follow the baseline method. Eligibility is unchanged when a subgoal already corresponds to an atomic action. Call scopes for the other frameworks are described in the experimental protocol.

The input retains the baseline method's message order, history truncation rules, and model chat template. We obtain final-goal positions I_G by case-insensitive literal matching of the complete final-goal text in the retained messages for the actual task. Fixed system rules and few-shot messages are excluded from matching. If the final goal appears multiple times in the actual task messages, including verbatim repetitions in the generated history, all matching positions are included in I_G. Goal readout and response-side masking use this same position set. Nonverbatim paraphrases of the goal remain in the ordinary context.

In this interface, the current subgoal is taken from the task node selected by the controller and localized in the most recent `Your current task:` field that matches its text. Its positions form I_g. We map message character spans to tokens using tokenizer offsets, including each valid token whose character span intersects the field span. All remaining prefix positions belong to I_B, including system rules, tool definitions, observations, history, and template markers outside the fields. The input neither appends an extra subgoal copy nor reorders the original messages.

The current assistant response-start marker belongs to I_E and is handled separately from the input prefix X_k. After locating the fields, we check that the goal and subgoal position sets are nonempty, disjoint, and entirely before the response-start marker. If localization or token mapping fails, the call uses the baseline method and records the fallback reason.

After successful localization, each subgoal position still follows the visibility rules of the base mask. If a position cannot attend to any goal position, its row in P is zero and its value remains unchanged. Response-side goal masking still applies to that call. For each call, we record which goal positions are visible to each subgoal position.

The original message layout also determines which observations can enter the relay. A new observation placed after all subgoal positions cannot affect prefix encoding at those positions, but response generation can still access it directly under the base visibility rules.

B.2 Relay Construction and Interaction Procedure

Each call using SR first completes native prefix encoding to obtain the full KV cache and the statistics needed for goal readout. Denote these statistics by $\mathcal{W}_k$, stored either as P or as the equivalent quantities $(c_j, \mathbf{t}_j)$. Let $\mathcal{T}_\rho$ denote relay construction in Eqs. (4)–(7). The two steps are

$$\begin{aligned}(\mathcal{C}_k^{\mathrm{full}}, \mathcal{W}_k) &= \mathrm{Prefill}_\theta(X_k; M^{\mathrm{base}}),\\ \mathcal{C}_{k,0}^{\mathrm{SR}} &= \mathcal{T}_\rho(\mathcal{C}_k^{\mathrm{full}}; \mathcal{W}_k).\end{aligned} \tag{B.1}$$

Every layer and chunk of prefix encoding reads values without transfer. The subgoal value cache is updated only after the entire prefix has been encoded. This write does not modify the hidden states computed for that prefix.

To enforce the access constraint from the prediction of the first output token, the model encodes the assistant response-start marker under the SR mask using the updated cache. Its final representation is then used to predict the first output token. Subsequent generation uses the same mask and appends response KV entries as tokens are generated. The prefix relay stays fixed within the call.

Algorithm B.1: Interaction with Subgoal Relay

Input: language model, final goal G, tool interface, baseline controller and message construction rules, task execution budget, and transfer budget ρ.

1. The controller determines the current subgoal and call state. If the eligibility conditions in Appendix B.1 hold, proceed to Step 2. Otherwise, generate a response using the baseline method and proceed to Step 6.
2. Construct the input prefix and locate the fields. If localization fails, record the reason, generate a response using the baseline method, and proceed to Step 6.
3. Encode the entire prefix under the base mask to obtain the KV cache and goal-readout statistics.
4. Construct the relay using Eqs. (4)–(7). Update the subgoal value cache once, retaining all keys and other prefix values.
5. Encode the response-start marker and generate the response under the SR mask, appending response cache entries as generation proceeds.
6. The controller parses the response, expands tasks or invokes tools, appends the response and feedback to the history, and releases the current cache. Return to Step 1 if task execution continues.

Output: task execution trajectory and terminal state.

The next call uses an input prefix containing the new response and tool feedback. Call eligibility and field locations are checked again. The relay is reconstructed from this prefix even if the subgoal text has not changed.

To reconstruct the cache for the same call, we fix its input prefix and existing response tokens, repeat base prefix encoding and value transfer, and then encode the existing response, including the response-start marker, under the SR mask. This procedure neither resamples the response nor replays tool actions. A single prefill over the concatenated prefix and response cannot replace the intermediate value update.

B.3 Attention Adaptation and Numerical Implementation

Native attention computation. SR follows the base model's projections, query and key normalization, positional encoding, and score computation order. The base mask includes the original causal, padding, and window constraints. A hard goal mask is added during response generation. Goal-readout weights come from the actual prefix attention distribution, retaining its normalization denominator over all valid keys.

The implementation can extract the required weights directly or accumulate only c_j and $\mathbf{t}_j$, without storing the full attention matrix. If the backend does not expose these statistics, we recompute attention scores for the subgoal query rows using the queries, keys, and positions from that prefix computation. Recomputation covers all valid keys rather than applying softmax only within goal positions.

Shared KV heads. In grouped-query attention, multiple query heads share a KV cache, so the shared values require a single update. Let $\mathcal{H}_u$ denote the set of query heads associated with KV head u. We compute each query head's attention distribution and then take their arithmetic mean:

$$P_{jr}^{(\ell,u)} = \frac{1}{|\mathcal{H}_u|} \sum_{h \in \mathcal{H}_u} A^{(\ell,h)} (M^{\text{base}})_{jr}. \tag{B.2}$$

We then use this mean distribution and the original values of value head u to compute c_j, $\mathbf{t}_j$, and the magnitude coefficients, and write a single shared update. The average is taken over post-softmax distributions, not queries or attention scores. The mean distribution retains nonnegative weights and an attention mass on the goal of at most one, so the convex combination and coefficient bound in Appendix A continue to hold.

Numerical precision and boundary cases. Value aggregation, norms, and ratios are computed in FP32, and updates are written in the cache's precision. If $\|\mathbf{d}_j\|_2 = 0$, the original value is retained. If $\|\mathbf{v}_{g,j}\|_2 = 0$ and $\|\mathbf{d}_j\|_2 > 0$, the update coefficient is zero. When $\rho = 0$, value writes are skipped.

The relative update bound is checked before writing to the cache. Numerical errors after conversion to cache precision are recorded separately. This distinguishes the exact-arithmetic relations in Appendix A from the actual changes in the low-precision cache.

Cache and parameter configuration. The implementation retains all cache entries and their original logical positions. Additional computation includes extracting or recomputing goal-readout weights, aggregating values, bounding update magnitudes, and writing to the cache.

The transfer budget ρ is selected on development data, then fixed and shared across all controlled layers and positions. The evaluation configuration and sensitivity analysis are reported separately. Comparison conditions use identical message construction, control rules, and decoding settings. Cost measurements include prefix encoding, relay construction, response generation, and other calls made during task execution.

C Experimental Protocol and Measurement Details

C.1 Implementation and Measurement Details

Backbone configurations. Qwen3 uses the original weights for each model size, and Llama uses the Instruct weights listed in the main text. Action limits for ALFWorld, ScienceWorld, and WebShop are 30, 100, and 16. Time limits are 300/540/225 seconds for Qwen3-32B, 180/324/135 seconds for Qwen3-14B, and 100/180/75 seconds for the remaining backbones. Budgets are fixed before testing and shared by all four frameworks and Base/SR within each backbone. Cross-backbone analyses compare paired changes within each configuration.

Configuration selection. The three exemplar sets comprise the original set and two alternatives drawn from development data. Temperature 0 uses greedy decoding. Other sampling settings, prompts, control procedures, input and generation lengths, truncation, and budgets are fixed. Ties favor the original configuration, then follow a predefined order. Test results are not used for selection. SR uses development-selected parameters, and design ablations fix the main parameters. Alternative control strategies use the same permitted development data. Soft selects a finite negative bias on the development set.

PASTA-g configuration. We adapt PASTA (Zhang et al., 2024) to emphasize the current subgoal and refer to this control as PASTA-g. The selected span contains the current subgoal text, excluding its field heading and repetitions in the history. Prefix encoding follows the base model. Attention steering applies only during response generation. PASTA-g retains the original final-goal input, leaves subgoal values unchanged, and does not apply SR's goal-access mask.

We use `scale_position="exclude"`. In the selected attention heads, a bias of $\log \alpha$ is added to positions outside the current subgoal before softmax. Smaller α gives the subgoal greater relative weight, while $\alpha = 1$ leaves attention unchanged. We select heads and steering strength on 30 ScienceWorld development tasks that do not overlap with the test set. Heads are ranked by changes in mean development score. We compare the top 8, 16, and 32 heads and search $\alpha \in \{0.01, 0.1, 0.3, 0.5, 1.0\}$. We select the configuration with the highest mean development score. Ties favor fewer heads, then α closer to 1. The selected configuration is fixed during testing. The layer and head indices and steering strength are saved with the evaluation configuration and corresponding results.

Base implementations. Plan-and-Execute is a controlled in-house implementation, ReCAP and HiAgent are adapted frameworks, and TDP is a reproduction from the paper. Each Base/+SR comparison uses the same implementation and environment interface, isolating the addition of SR within that framework. Tasks with missing fields, fallback, or zero updates remain in the main-result denominator.

Framework interfaces. We enable SR after the framework has determined the current subgoal. The adapter locates the final goal and current subgoal in the actual input messages or controller fields and maps them to input token positions. It preserves message order without adding a copy of the subgoal. Table C.1 summarizes the intervention scope. Eligible calls use the cache procedure in Appendix B.2.

Before each intervention, we check that the goal and subgoal spans are nonempty, do not overlap, and precede the response. If localization fails, the call follows the original framework, and its task remains in the evaluation. Base and Base+SR use the same backbone, prompt templates, tool interfaces, history management, and execution budgets. Full disablement skips both value transfer and the access mask. A zero transfer bound with the mask retained gives Mask-Only. Each run configuration records the model and code revisions, task list, exemplar set, decoding settings, context and generation limits, and ρ. Field-localization rules and prompt templates are saved with these records. PASTA-g records additionally contain the selected heads and steering strength.

Table C.1: Framework interfaces for the main evaluation. SR intervenes after the current subgoal is determined. Higher-level control procedures retain their original behavior.

Framework	Base implementation	Intervention scope
Plan-and-Execute	Controlled in-house implementation	Local executor calls after the planner determines the current subtask. Global planning and replanning are unchanged.
ReCAP	Adapted recursive planning framework	Non-root downward calls for subtask refinement or action generation. Root planning and upward revision retain baseline access.
HiAgent	Adapted subgoal-memory framework	Action calls after the current subgoal is selected. Subgoal selection and switching, memory compression, and retrieval are unchanged.
TDP	Reproduction from the paper	Action decisions for the task assigned by the current local plan. High-level task planning and scheduling are unchanged.

Compute configuration. Experiments use 8 NVIDIA A800 GPUs. Each Qwen3-8B run uses one GPU with BF16 precision and batch size 1. Multiple GPUs run independent tasks in parallel. Execution within each task is serial, without cross-task batching or KV-cache reuse. Paired conditions share the software environment and use SDPA math with explicit masking support. Full/FlashAttention-2 is used in the fixed-workload microbenchmark (Table D.8), while the main table compares methods under the same backend. Appendix D.12 derives backend break-even points from the cost breakdown. Timing starts when a task is received and ends at success, failure, or a budget stop. It includes model computation, tools, and waiting, but excludes model loading, environment initialization/reset, and offline construction. After 10 warm-up runs on development tasks, conditions are randomly interleaved across five seeds. The fixed-string microbenchmark uses 50 replays after warm-up.

C.2 Setup for the Motivation Experiments

The motivation experiments distinguish whole-task goal visibility from the use of goal information in textual handoff. Neither diagnostic uses the explicit value transfer of SR. Figure 1(a) reports whole-task completion. Panels (b,c) examine one matched local entry per source task.

Task subset and configuration. We sample 60 tasks from the ALFWorld `train` split, stratified by task type with sampling seed 42. The subset excludes tasks used for prompt examples, configuration selection, or SR parameter selection. ReCAP uses Qwen3-4B with frozen weights, fixed prompt

examples, greedy decoding, and thinking disabled. Each condition is evaluated once per task. Whole-task action and time limits match the main Qwen3-4B configuration: 30 environment actions and 100 seconds. Other generation and controller settings are shared across the paired conditions. The main evaluation instead uses 134 `valid_unseen` tasks and development-selected configurations averaged over five evaluations.

Whole-task goal visibility. Full retains the original final-goal field during local execution, while Hide removes it before input encoding. High-level planning and upward feedback-based revision retain access to the final goal in both conditions. Subgoal selection, history management, tool interfaces, and context construction follow the same rules. Task completion is determined by the environment evaluator. Failed tasks and tasks that exhaust their budgets remain in the denominator. Full succeeds on 27 of 60 tasks (45.0%), while Hide succeeds on 29 (48.3%), a gain of 3.3 percentage points before rounding.

Textual handoff conditions. The same Qwen3-4B model generates local explanations with and without the final goal. We give each explanation and the current subgoal to the executor, hiding the final goal during execution. We then reuse exactly the same goal-conditioned explanation and additionally provide the final goal during execution. All three conditions start independently from the same entry state, history, and subgoal, using the same local budgets. Table C.2 summarizes the conditions and completion counts.

Table C.2: Textual-handoff conditions on 60 matched entries, one per source task and one execution per condition. All rows include a local explanation.

Explanation generation	Goal given to executor	Completed entries	Completion (%)
Without the final goal	No	34/60	56.7
With the final goal	No	38/60	63.3
With the final goal (same explanation)	Yes	35/60	58.3

Entry states, evaluation, and stopping. For each source task, we select the first restorable non-root entry whose current subgoal differs from the root task, using a fixed reference execution configuration. Entries are selected before running the handoff comparisons. Each condition starts independently from the same environment state, history, and current subgoal, with a limit of 10 environment operations and 40 seconds, whichever is reached first. The timer covers model inference, controller processing, and environment interaction, excluding entry restoration and prior explanation generation. Each operation submitted to the environment counts toward the action limit, including observation queries and invalid operations.

We define completion predicates from the original subgoal before comparison and share them across conditions. After each operation, we check the environment state and stop at the first successful completion. When the environment terminates, control is returned, or a budget is exhausted before the predicates are satisfied, the run counts as non-completion. A model's completion statement does not establish success. Generation failures and invalid outputs remain in the denominator. This local stopping rule does not change the whole-task controller or the end-to-end timing protocol used in the main experiments.

Aggregation and paired changes. Each of the 60 tasks contributes one binary outcome per condition, so equal task weights reduce to counts divided by 60. The first handoff comparison changes 2 entries from success to failure and 6 from failure to success, yielding a net gain of 4 entries (6.7 percentage points). Providing the goal during execution with the explanation fixed changes 5 entries from success to failure and 2 in the reverse direction, yielding a net decrease of 3 entries (5.0 percentage points). Table C.3 uses the same paired entries. All percentage-point differences are computed before rounding.

Stage deviation. With the same goal-conditioned explanation, 6 of 60 local executions (10.0%) show at least one unnecessary move to another stage when the goal is hidden, compared with 9 (15.0%) when it is provided. Necessary preparation, appropriate return of control, and authorized

Table C.3: Paired outcome changes for the two local comparisons. Counts use the same 60 entries as Table C.2. Net change equals failure-to-success minus success-to-failure, divided by 60. pp denotes percentage points.

Comparison	Success to failure	Failure to success	Net change (pp)
Use goal to generate explanation	2	6	+6.7
Also provide goal during execution	5	2	−5.0

plan revisions are excluded. Semantic labels use the model-assisted annotation and human review procedure in Appendix E.3. The lower completion rate is accompanied by more stage deviations.

D Supplementary Experimental Results

D.1 Evaluation across Backbones

Tables D.1–D.2 report results for Qwen3-4B, Qwen3-14B, Qwen3-32B, Llama-3.1-8B-Instruct, and Llama-3.2-3B-Instruct across four agent frameworks and three environments. Within each pair, Base and Base+SR share model weights, framework settings, and execution budgets. All model roles use the same backbone. Built-in thinking is disabled for Qwen3, and each framework retains its original reasoning procedure. Backbone-specific budgets are given in Appendix C.1.

Table D.1: Paired Base/+SR results across backbones (1/2). ALFWorld reports success rate (%), ScienceWorld reports mean final score, and WebShop reports mean purchase reward multiplied by 100. Higher is better. Results are means ± sample standard deviations over five complete evaluations. Each pair shares the model, framework, and budget. Qwen3-8B results appear in Table 1.

Method	Variant	ALFWorld	ScienceWorld	WebShop
		Qwen3-32B		
Plan-and-Execute	Base	75.1 ± 2.7	28.0 ± 3.4	48.4 ± 1.7
	+SR	78.2 ± 2.4	32.3 ± 3.2	53.5 ± 1.7
ReCAP	Base	82.1 ± 1.4	36.4 ± 3.5	55.2 ± 1.1
	+SR	84.5 ± 1.7	41.4 ± 3.6	62.2 ± 0.8
HiAgent	Base	75.4 ± 3.9	34.3 ± 3.0	53.0 ± 2.6
	+SR	77.2 ± 2.6	38.7 ± 2.8	57.2 ± 2.4
TDP	Base	82.7 ± 2.8	39.9 ± 2.0	51.9 ± 2.2
	+SR	83.9 ± 1.9	44.8 ± 1.6	57.4 ± 2.4
		Qwen3-14B		
Plan-and-Execute	Base	70.3 ± 6.1	27.2 ± 3.1	44.7 ± 2.2
	+SR	73.3 ± 4.8	30.7 ± 3.0	49.1 ± 2.4
ReCAP	Base	78.8 ± 2.9	34.7 ± 3.0	52.5 ± 1.7
	+SR	82.1 ± 2.4	39.2 ± 2.7	58.1 ± 1.6
HiAgent	Base	64.0 ± 2.1	32.1 ± 2.6	49.8 ± 2.1
	+SR	68.2 ± 3.1	35.6 ± 2.5	53.8 ± 2.1
TDP	Base	75.4 ± 2.8	37.8 ± 2.6	51.1 ± 2.6
	+SR	77.0 ± 3.0	41.4 ± 2.3	56.6 ± 3.1

SR improves the mean task metric on all five additional backbones, covering different model sizes and two model families. For ReCAP on ScienceWorld, Qwen3-32B improves from 36.4 to 41.4, while Llama-3.2-3B-Instruct improves from 12.0 to 15.4. The gains thus extend beyond Qwen3-8B to backbones with different baseline performance levels.

Table D.2: Paired Base/+SR results across backbones (2/2). ALFWorld reports success rate (%), ScienceWorld reports mean final score, and WebShop reports mean purchase reward multiplied by 100. Higher is better. Results are means ± sample standard deviations over five complete evaluations. Each pair shares the model, framework, and budget. Qwen3-8B results appear in Table 1.

Method	Variant	ALFWorld	ScienceWorld	WebShop
Qwen3-4B				
Plan-and-Execute	Base	51.3 ± 2.3	17.1 ± 2.0	35.6 ± 1.7
	+SR	56.0 ± 2.2	19.8 ± 1.7	42.6 ± 1.8
ReCAP	Base	63.1 ± 3.7	20.1 ± 2.3	42.8 ± 1.6
	+SR	69.0 ± 3.4	23.4 ± 2.3	51.8 ± 1.5
HiAgent	Base	53.3 ± 4.4	18.9 ± 2.4	38.9 ± 2.3
	+SR	57.3 ± 2.8	21.4 ± 3.1	44.4 ± 2.4
TDP	Base	64.0 ± 4.4	23.1 ± 2.3	40.5 ± 2.2
	+SR	66.3 ± 3.9	26.1 ± 2.4	47.7 ± 2.0
Llama-3.1-8B-Instruct				
Plan-and-Execute	Base	51.2 ± 5.3	17.7 ± 2.2	31.0 ± 0.9
	+SR	57.3 ± 3.8	21.9 ± 2.3	37.1 ± 0.4
ReCAP	Base	62.5 ± 3.6	22.0 ± 2.4	38.6 ± 1.6
	+SR	66.6 ± 5.5	26.7 ± 2.8	47.2 ± 1.8
HiAgent	Base	53.1 ± 4.5	23.5 ± 2.5	37.4 ± 2.1
	+SR	56.0 ± 4.7	27.9 ± 2.8	42.4 ± 2.2
TDP	Base	62.1 ± 6.4	25.6 ± 2.0	35.8 ± 1.5
	+SR	65.2 ± 6.3	30.1 ± 1.8	41.9 ± 1.8
Llama-3.2-3B-Instruct				
Plan-and-Execute	Base	33.6 ± 3.3	9.8 ± 1.2	29.2 ± 1.6
	+SR	41.0 ± 5.1	12.8 ± 1.3	35.3 ± 1.5
ReCAP	Base	46.6 ± 1.9	12.0 ± 1.7	34.6 ± 1.7
	+SR	52.2 ± 1.3	15.4 ± 1.9	42.5 ± 1.9
HiAgent	Base	36.9 ± 4.2	12.7 ± 1.5	34.8 ± 1.6
	+SR	40.6 ± 4.4	16.0 ± 1.3	39.7 ± 1.4
TDP	Base	43.6 ± 2.9	13.9 ± 1.0	33.3 ± 1.9
	+SR	48.2 ± 3.9	17.2 ± 1.2	39.5 ± 1.7

D.2 Distribution of Gains across Tasks

Table D.3: Task-level gains and regressions for ReCAP with Qwen3-8B. Scores are first averaged over five repetitions per task, then classified by positive, zero, or negative SR−Full differences. The respective counts are 129/46/36 for ScienceWorld and 280/145/75 for WebShop. Percentages use all tasks as the denominator and are not success rates. Mean gains and losses are computed within the positive- and negative-difference groups, with losses reported as positive magnitudes. Overall mean gains are task-averaged differences. Weighting by unrounded group proportions recovers the main results. Score differences are in points.

Environment	Improved (%)	Unchanged (%)	Degraded (%)	Mean gain if improved	Mean loss if degraded	Overall mean gain
ScienceWorld	61.1	21.8	17.1	8.89	4.90	4.6
WebShop	56.0	29.0	15.0	17.04	5.60	8.7

Table D.3 decomposes overall score changes into their coverage and magnitude. Scores improve on 61.1% of ScienceWorld tasks and 56.0% of WebShop tasks, covering a majority of the evaluation tasks. Mean gains on improved tasks also exceed mean losses on degraded tasks. On ScienceWorld, these values are 8.89 and 4.90 points, respectively, with the same relationship on WebShop. The net gains in the main table therefore reflect both broad coverage and larger average improvements.

Scores still decrease on 17.1% and 15.0% of tasks in the two environments, respectively, indicating that changes to goal guidance affect tasks differently. Combining positive and negative changes gives overall mean gains of 4.6 points on ScienceWorld and 8.7 points on WebShop. This distribution complements the aggregate means: SR improves many tasks, while the specific gain depends on how each task is executed.

D.3 Component Ablations across Frameworks

Table D.4: Component ablations across frameworks on ScienceWorld with Qwen3-8B. Score is the mean final score, with higher values better. Full applies no intervention, Mask-Only masks access during response generation, Transfer-Only transfers values, and SR combines both. Each framework retains its original input construction and main parameters. ReCAP reuses Table 2, and HiAgent Full/SR reuse Table 1.

Condition	ReCAP Score	HiAgent Score
Full	28.4	24.9
Mask-Only	30.1	26.3
Transfer-Only	29.3	25.8
SR	33.0	28.8

Table D.4 compares component ablations for ReCAP and HiAgent on ScienceWorld, both using Qwen3-8B. In HiAgent, full SR scores 28.8. Removing value transfer and retaining response-side masking lowers the score to 26.3. Removing masking and retaining transfer lowers it to 25.8. Both single-component variants outperform Full but remain below the complete method, matching the direction observed for ReCAP. Both components contribute even with HiAgent's existing subgoal-based memory management.

D.4 Goal Control Strategies and Transfer Design

Table D.5: Goal control strategies and SR transfer designs on ScienceWorld with ReCAP and Qwen3-8B. Score is the mean final score, with higher values better. Prompt adds only a stage instruction. GM masks goal access from the encoding stage. Soft partially suppresses access during response generation. PASTA-g emphasizes the current subgoal. SR-Uniform preserves total attention weight on the goal, distributes it uniformly within the goal, and retains clipping. SR-NoClip removes update clipping. PASTA-g adapts the published PASTA method(Zhang et al., 2024) to our tasks and does not report the original paper's results. Overlapping conditions reuse the main-text means.

Condition	Score ↑
Goal control strategies	
Full	28.4
Prompt	30.2
GM	29.4
Soft	30.7
PASTA-g	31.6
SR transfer design ablations	
Mask-Only	30.1
SR-Uniform	31.9
SR-NoClip	31.3
SR	33.0

Table D.5 compares goal control strategies and SR transfer designs on ScienceWorld using ReCAP with Qwen3-8B. Prompt, Soft, and PASTA-g all have higher mean scores than Full. These controls act through stage instructions, reduced goal access, or subgoal emphasis. SR combines subgoal-conditioned value transfer with goal-access masking and has the highest mean score in this comparison, 33.0. In contrast, GM, which masks goal access from the encoding stage, scores 29.4. GM changes access to goal information at both subgoal and other prefix positions. Different masking stages therefore retain different information, and these strategies cannot be characterized solely by whether the overall goal is masked.

SR-Uniform preserves total attention weight on the goal but distributes it uniformly within the goal, yielding a mean score of 31.9. SR-NoClip uses the original allocation without clipping and scores 31.3. Both means are below SR's 33.0. These comparisons favor the original allocation and bounded updates, motivating the parameter analysis and per-row norm-matching control below.

D.5 Parameter Sensitivity and Transfer across Frameworks

We control value-transfer strength through the relative update bound ρ. Setting $\rho = 0$ disables transfer and corresponds to Mask-Only. Larger ρ permits larger relative updates. ReCAP uses $\rho = 0.20$, selected on development data and fixed before testing.

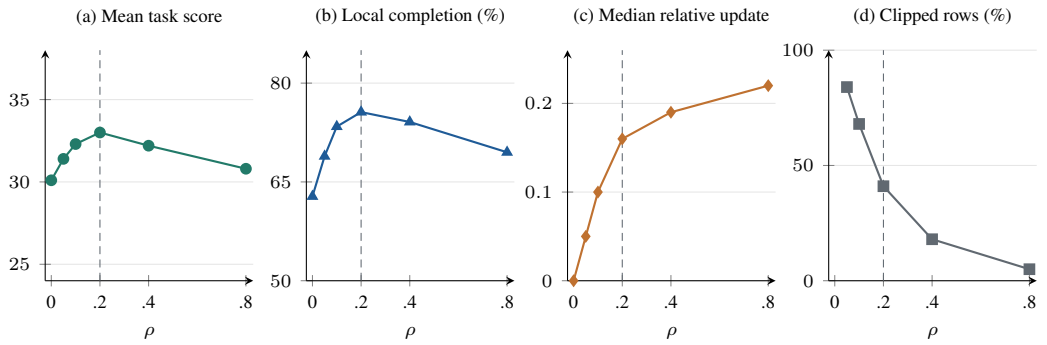

Figure D.1: Sensitivity to transfer strength on ScienceWorld with ReCAP and Qwen3-8B. ρ bounds the relative value-vector update, and the dashed line marks the main setting of 0.20. (a) Mean task score. (b) Subgoal completion rate from fixed diagnostic entry states. (c) Median relative update after writeback. (d) Fraction of nonzero candidate-update rows that trigger clipping. Panels (c,d) pool eligible rows from complete trajectories run separately at each ρ. A row is indexed by call, layer, KV head, and subgoal position. $\rho = 0$ skips transfer and corresponds to Mask-Only. Panel (d) omits the zero point. Error bars are omitted.

For panels (c,d), each ρ starts from the same task initial conditions and generates its own trajectories. Panel (c) pools rows with nonzero original value norms, including zero updates. Rows with zero original norms are counted separately and excluded from this ratio. Panel (d) pools rows with nonzero candidate updates. The row populations can differ as trajectories diverge.

Figure D.1 shows that moderately increasing transfer strength improves local execution and task performance. As ρ increases from 0 to 0.20, mean task score rises from 30.1 to 33.0 and local completion rate from 62.8% to 75.6%. Scores remain above the no-transfer condition throughout 0.10–0.40, so the gains are not confined to one isolated setting. Increasing the bound to 0.80 further enlarges the actual relative update and reduces the fraction of clipped rows, but both task score and local completion decline. Larger updates therefore do not continually improve execution, supporting a bound on the magnitude of injected goal guidance.

Table D.6: Transfer of the relay parameter to HiAgent on ScienceWorld with Qwen3-8B. ρ is the relative update bound. Framework-specific selection uses HiAgent development data, while direct transfer uses ReCAP's 0.2. Both are fixed before testing. Score is the mean final score, and T_{end} is mean end-to-end time per task in seconds. Conditions share tasks, seeds, and budgets. Full and framework-selected SR reuse the main table.

HiAgent condition	ρ	Score ↑	T_{end} (s) ↓
Full	N/A	24.9	88.2
SR, framework-selected	0.1	28.8	86.0
SR, directly transferred	0.2	28.1	87.1

Table D.6 examines reuse of the transfer setting. Applying ReCAP's $\rho = 0.20$ directly to HiAgent improves the score from Full's 24.9 to 28.1. Selecting $\rho = 0.10$ on HiAgent development data further improves it to 28.8. Direct transfer retains much of the gain, with framework-specific selection providing a further improvement.

D.6 Budget-Constrained Evaluation on ALFWorld

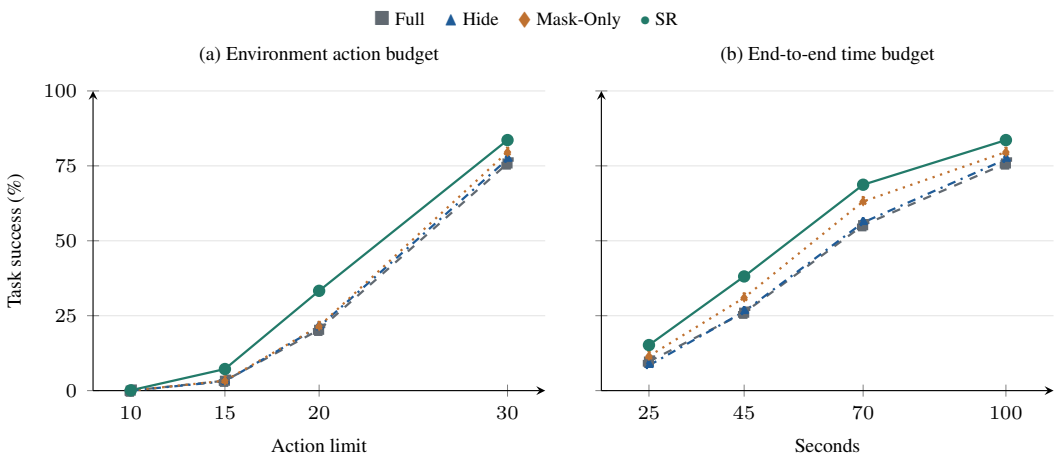

Figure D.2: Success rates under matched budgets on ALFWorld unseen for ReCAP with Qwen3-8B. (a) Action-limit sweep with a fixed time limit of 100 seconds. (b) End-to-end time-limit sweep with a fixed action limit of 30. All four curves use the same 134 tasks. The same task records are truncated at action or time boundaries. Maximum-budget points match Table 2. Time includes all online model computation, SR computation, retrieval and control, and environment interaction.

Figure D.2 compares task success on ALFWorld under action and time budgets. At an action limit of 20, SR achieves 33.3% success, compared with 20.3% for Full and 21.5% for Mask-Only.

Time-budget comparisons show the same direction. Including transfer computation and all online overhead, SR achieves 68.7% success within 70 seconds, above Full's 55.2%. SR therefore completes more tasks within the same action and time limits.

D.7 Transfer Direction at Matched Magnitudes

Both matched conditions start from the same diagnostic entry states. At each eligible call, we compute the original-weight and uniform-weight candidates from that condition's current unmodified prefix cache. Their smaller post-clipping norm defines a common radius for each row. Orig-Matched writes the original direction at this radius, while Uniform-Matched writes the uniform direction. If either candidate is zero, the common radius is zero, and the entry remains in the evaluation. Response-side masking and execution settings are unchanged. The radius is matched between candidates within each prefix. Subsequent trajectories may diverge and yield different radii.

Table D.7: Transfer directions at matched magnitudes on ScienceWorld with ReCAP and Qwen3-8B. Local completion uses subgoal-state criteria. Constraint violations measure violations of persistent requirements during local execution, distinct from first-call boundary violations in Figure 3(a). Both are percentages, reported as means ± sample standard deviations over five seeds. The Matched variants scale the original/uniform directions to a common per-row radius and retain all other settings. Local completion uses the entry states in Figure D.1(b): 80 source tasks and 320 entry states, paired by source task.

Condition	Local completion (%) ↑	Constraint violations (%) ↓	Update magnitude
Mask-Only	62.8 ± 2.7	27.6 ± 2.3	0
SR	75.6 ± 2.2	16.9 ± 1.8	Original, clipped
SR-Uniform	70.8 ± 2.5	21.8 ± 2.1	Uniform, clipped
Orig-Matched	73.9 ± 2.4	17.5 ± 1.9	Matched per-row radius
Uniform-Matched	68.4 ± 2.8	22.6 ± 2.2	Matched per-row radius

Under the per-row norm-matching rule, Orig-Matched achieves 73.9% mean local completion, above Uniform-Matched's 68.4%, while constraint violations decrease from 22.6% to 17.5% (Table D.7). The original allocation thus retains an advantage under this magnitude control. Orig-Matched also remains below SR's 75.6% completion, indicating that shrinking updates to the common radii changes the outcome. These results favor subgoal-conditioned allocation together with the bounded update used by SR.

D.8 Computational Overhead and Total Task Cost

Table D.8: Mean per-call computational overhead for Qwen3-8B under a fixed workload: batch size 1, a 4096-token prefix with a 128-token goal field, and 64 decoded tokens. Full (native) uses the deployment backend. Full (matched) shares the backend with Mask-Only and SR. Hardware and precision are identical. Times are in milliseconds and memory in MiB. Transfer includes weight retrieval, aggregation, clipping, and writeback. Full replay is the sum of the first three rows. SR retains the complete KV cache. Fixed-length per-call time does not directly translate into task-level time in the main table.

Measurement	Full (native)	Full (matched)	Mask-Only	SR
Prefix encoding	84.6	93.1	93.1	93.1
Weight retrieval and transfer	N/A	N/A	N/A	61.4
Response marker and fixed-string decoding	1246.8	1284.2	1288.6	1290.1
Full replay	1331.4	1377.3	1381.7	1444.6
Peak memory	17248	17632	17680	18096

The fixed-workload microbenchmark in Table D.8 shows that SR's additional computation is concentrated in weight retrieval and transfer. Under the same backend, Full and SR both take 93.1 milliseconds to encode the prefix. SR adds 61.4 milliseconds for weight retrieval and transfer, with only a small increase in decoding time. Full replay increases from 1377.3 to 1444.6 milliseconds, approximately 4.9%, while peak memory increases by approximately 2.6%. Relative to Full on the native efficient backend, full replay takes approximately 8.5% longer. These results characterize the per-call costs of transfer and backend choice at fixed input and output lengths.

Table D.9: Per-task workload and end-to-end cost breakdown for ReCAP with Qwen3-8B. Input counts sum the prefixes actually processed across calls, and output counts include all model roles. Tokens are in thousands and times in seconds. All task attempts are included. T_{model} includes SR computation. Online processing covers non-model retrieval, memory, parsing, and control. Environment interaction includes tool execution and waiting. These three disjoint components sum to T_{end}, with possible 0.1-second differences after independent rounding. Its mean $\pm$ sample standard deviation over five evaluations reuses the main table. Model loading, environment initialization/reset, and offline construction are excluded.

Environment	Condition	Input	Output	T_{model}	Online processing	Environment interaction	T_{end}
ALFWorld	Full	65.43	1.96	41.3	3.8	8.8	53.9 ± 1.6
	SR	55.84	1.72	36.4	3.3	8.2	47.9 ± 1.4
ScienceWorld	Full	152.63	4.08	85.5	8.9	12.9	107.3 ± 3.8
	SR	132.65	3.75	79.0	8.5	12.7	100.2 ± 3.5
WebShop	Full	43.35	0.88	18.6	2.0	9.9	30.5 ± 1.4
	SR	36.98	0.80	16.5	1.8	9.5	27.8 ± 1.3

Total task cost also depends on call count, context length, and environment interaction. The ReCAP/Qwen3-8B breakdown shows that SR reduces total input and output tokens in all three environments. Together with fewer calls and largely unchanged output length per call in the main text, this is consistent with reducing repeated decisions and recovery work. Additional per-call

computation can therefore coexist with a lower total workload, and both determine the final execution time.

In all three environments, reduced ordinary model computation offsets transfer overhead, while changes in online processing and environment interaction time are smaller. Together with lower total input and output, the net model-time savings are consistent with fewer repeated calls and less processing of accumulated history. Fixed-workload calls still incur overhead, but the task requires less computation overall, allowing task performance and total execution time to improve together.

D.9 Effect of Base Configuration on SR Gains

We compare a fixed reference configuration with development-selected configurations to examine how the base configuration affects SR gains. Each framework–environment pair has seven candidates: a fixed reference with temperature 0.7, top-p 0.8, and top-k 20, plus six configurations combining three exemplar sets with temperatures 0 and 0.2. ALFWorld, ScienceWorld, and WebShop use 20, 24, and 50 development tasks, respectively, disjoint from the test sets, with three repetitions per task. Selection uses only Base's mean development task metric. The selected configuration is then applied to both Base and Base+SR, which share test tasks, execution budgets, and framework control procedures.

Table D.10: Base/+SR under the fixed reference configuration with Qwen3-8B. Metrics and test sets match Table 1, and only means are shown. Each environment's two columns share the fixed reference configuration without development selection. The main table reports development-selected configurations on the same test tasks and budgets.

	ALFWorld		ScienceWorld		WebShop	
Framework	Base	+SR	Base	+SR	Base	+SR
Plan-and-Execute	58.2	65.2	19.4	23.8	36.3	43.5
ReCAP	72.4	80.6	24.8	30.1	44.2	53.3
HiAgent	56.7	63.0	22.6	26.8	40.6	46.6
TDP	68.7	75.5	29.2	33.9	42.4	50.4

Development selection improves base-agent performance while retaining the additional gains from SR. For ReCAP on ScienceWorld, Base and SR score 24.8 and 30.1 under the fixed reference configuration. Development selection raises these scores to 28.4 and 33.0. As the base improves, the SR gain narrows from 5.3 to 4.6 points but remains positive. All four frameworks show positive mean gains in all three environments under both configurations, supporting the complementary value of configuration selection and changes to goal guidance.

D.10 Component Effects and Paired Intervals

Table D.11: Paired component effects for ReCAP with 95% intervals. In the upper group, ALFWorld uses percentage points and the other environments use score points. The lower group reports time differences in seconds. The four conditions retain original input construction and run independently from matched initial states. Masking effects without/with transfer are Mask-Only−Full and SR−Transfer-Only. Transfer effects without/with masking are Transfer-Only−Full and SR−Mask-Only. The interaction $(\mathrm{SR} - \mathrm{Mask}) - (\mathrm{Transfer} - \mathrm{Full})$ describes non-additivity on the score scale. SR−TH compares handoff strategies. Intervals follow Appendix D.11. Independent rounding can make differences deviate by 0.1 from subtraction of displayed means.

Paired effect	ALFWorld	ScienceWorld	WebShop
Masking (without transfer)	3.7 [-0.7, 8.1]	1.7 [-1.5, 4.3]	4.5 [2.9, 6.1]
Masking (with transfer)	6.0 [1.3, 10.6]	3.7 [0.1, 7.0]	6.7 [5.4, 8.0]
Transfer (without masking)	1.8 [-2.5, 6.0]	0.9 [-1.2, 3.0]	2.0 [0.6, 3.4]
Transfer (with masking)	4.0 [-0.3, 8.8]	2.9 [-0.4, 6.4]	4.2 [2.3, 6.0]
Interaction	2.2 [-4.2, 9.0]	2.0 [-2.7, 7.0]	2.2 [0.0, 4.3]
Handoff strategies: task-metric differences			
Handoff: SR−TH	3.0 [-1.2, 7.3]	1.6 [-2.2, 5.2]	2.6 [0.8, 4.3]
End-to-end time differences (s, negative means savings)			
SR−Mask-Only	-2.5 [-4.3, -0.8]	-3.4 [-7.8, 0.6]	-1.3 [-2.5, -0.2]
SR−TH	-16.8 [-18.4, -15.2]	-34.6 [-37.4, -31.8]	-8.8 [-10.2, -7.6]

Paired differences further reveal how masking and transfer depend on each other. On ScienceWorld, value transfer adds 0.9 points on average without response-side masking and 2.9 points with masking. ALFWorld and WebShop show the same relationship between means, consistent with jointly retaining guidance through transfer and restricting direct goal access during response generation. Mean interactions are positive in all three environments, but their intervals include or approach zero. Combining the components gives higher mean task performance, while the current uncertainty does not clearly distinguish non-additive gains from additive contributions.

Compared with textual handoff, SR has a positive mean task-metric difference in all three environments. The interval is above zero on WebShop and crosses zero on ALFWorld and ScienceWorld. Time-difference intervals are negative in all three environments, consistent with internal transfer avoiding additional instruction generation. SR also has lower mean execution time than Mask-Only. The intervals are below zero on ALFWorld and WebShop but cross zero on ScienceWorld. The complete method's cost advantage therefore depends on whether retained guidance reduces subsequent work, rather than on transfer itself being cheaper.

D.11 Paired Analysis of Gains across Frameworks

Paired statistics use arrays of task results over five repetitions, with crossed resampling of tasks and seeds. Each resample draws both axes independently with replacement and uses their product weights. Compared conditions share the resampling weights to preserve task and repetition pairing. Intervals use the 2.5% and 97.5% quantiles of differences from 10,000 resamples. This crossed resampling procedure follows Owen (2007).

Table D.12: SR−Base task-metric and time differences with 95% intervals for each framework. ALFWorld task-metric differences are percentage points, and the others are score points. Time differences are seconds, with negative values indicating savings. There are 134, 211, and 500 tasks in the respective environments, each repeated 5 times. Intervals use crossed paired resampling over tasks and seeds, as described in this subsection.

Framework	Environment	Task-metric difference [95% interval]	Time difference [95% interval]
Plan-and-Execute	ALFWorld	6.6 [3.0, 10.6]	-4.1 [-5.3, -2.9]
	ScienceWorld	3.8 [1.4, 6.2]	-4.8 [-7.7, -1.8]
	WebShop	6.4 [4.7, 8.2]	-1.9 [-2.5, -1.3]
ReCAP	ALFWorld	7.8 [3.4, 12.2]	-6.0 [-7.4, -4.5]
	ScienceWorld	4.6 [2.0, 7.1]	-7.1 [-11.1, -3.6]
	WebShop	8.7 [7.1, 10.3]	-2.7 [-3.9, -1.6]
HiAgent	ALFWorld	5.7 [1.3, 10.3]	-3.0 [-4.3, -1.8]
	ScienceWorld	3.9 [1.3, 6.6]	-2.2 [-6.1, 2.3]
	WebShop	5.4 [3.6, 7.0]	-0.9 [-1.7, -0.1]
TDP	ALFWorld	5.5 [1.2, 10.1]	-4.2 [-5.7, -2.7]
	ScienceWorld	4.2 [2.0, 6.5]	-8.1 [-12.5, -3.7]
	WebShop	7.3 [5.6, 9.1]	-3.0 [-4.0, -2.1]

Paired results across frameworks complement the mean gains in the main table. Task-metric difference intervals are above zero for all four frameworks in all three environments, indicating that positive changes extend across planning and context-management approaches. ALFWorld intervals are wider, reflecting variability in success-rate differences with a limited task set. By retaining task and repetition correspondence, paired comparisons show the stability of gains beyond aggregate means.

Mean time differences are negative in every setting, and most intervals are also below zero, supporting task gains accompanied by net time savings. HiAgent's ScienceWorld interval still crosses zero, and its WebShop upper bound is close to zero.

D.12 Backend Acceleration and the Break-Even Point

We further examine how accelerating the base backend affects SR's net time savings. Holding task trajectories, quality, and non-model overhead fixed, let r be the fractional reduction in Full's model computation time, while SR's time remains unchanged. The time difference is

$$\Delta T(r) = T_{\text{Full}} - rM_{\text{Full}} - T_{\text{SR}}$$

where M_{Full} is Full's model computation time. $\Delta T(r) > 0$ means SR remains faster. Setting the difference to zero gives the break-even point $r^* = (T_{\text{Full}} - T_{\text{SR}})/M_{\text{Full}}$.

Table D.13: Backend break-even analysis for ReCAP with Qwen3-8B. The first three numeric columns are seconds and reuse Table D.9. The critical reduction r^* is computed from unrounded means. Only Full's model time is reduced. Quality, non-model overhead, and SR time remain unchanged. This is an analytical cost-sensitivity calculation.

Environment	Full total time	SR total time	Full model time	r^* (%)
ALFWorld	53.9	47.9	41.3	14.5
ScienceWorld	107.3	100.2	85.5	8.3
WebShop	30.5	27.8	18.6	14.5

Table D.13 shows that Full's model time must decrease by 14.5%, 8.3%, and 14.5%, respectively, to offset SR's total time savings in the three environments. ScienceWorld has a lower threshold, indicating greater sensitivity to the speed of the base backend. This analysis links efficiency gains to workload reduction: fewer calls and less context processing create room for time savings, while the savings retained in deployment depend on the execution costs of both backends.

D.13 Task Length and Workload Details

We group tasks by the number of environment operations in independently prepared reference trajectories verified as successful by the environment evaluator. Operation counts follow the same rules as evaluation. ALFWorld groups are short ($L \leq 10$), medium ($11 \leq L \leq 20$), and long ($L > 20$). ScienceWorld uses cutoffs of 30 and 60 operations. Every condition uses the same task groups, independently of its own execution length or measured improvement.

Task-horizon gains and rework changes for ReCAP with Qwen3-8B. Panels (a,b): short/medium/long groups use independent reference action counts. ScienceWorld contains 42/106/63 tasks, with score differences in points. ALFWorld contains 27/74/33 tasks, with success-rate differences in percentage points. The weighted overall gains are 4.6 points and approximately 7.8 percentage points, respectively. The units differ, so the absolute heights of the two curves should not be compared. Panel (c): task counts with decreased/unchanged/increased rework, classified by the SR−Full difference in mean rework actions per task. The denominators are 134, 211, and 500 for the three environments, including failures, without horizon grouping. Intervals are omitted.

Table D.14: Exact rework-change counts underlying Figure 4(c). Each task is classified by its mean SR−Full rework difference over repetitions.

Environment	Less	Same	More
ALFWorld	121	2	11
ScienceWorld	191	5	15
WebShop	415	17	68

Table 3 examines the computational costs associated with these execution changes. With SR, mean rework actions and model calls decrease in all three environments, while output length per call remains largely unchanged. The reduction in total output therefore primarily reflects fewer calls, consistent with avoiding repeated decisions and recovery work. Rework here includes actions that correct existing deviations or redo work that failed acceptance checks. It excludes normal exploration and planned stage transitions.

The lower workload also translates into net model-time savings. On ScienceWorld, total model calls decrease from 89.1 to 82.3. Even with 3.1 seconds of transfer computation, model time decreases from 85.5 to 79.0 seconds. ALFWorld and WebShop show the same direction. Together with the largely unchanged output length per call, these changes support savings from fewer repeated calls and less accumulated context processing, rather than faster individual inference calls.

D.14 Task Performance under Resource Budgets

These curves measure rewards reached by a cutoff along recorded executions. They do not rerun the controller with a new budget or measure adaptation to an announced deadline.

Figure D.3 compares task performance under action and time budgets. Under action limits, SR achieves higher scores at every evaluated ScienceWorld budget. On WebShop, its advantage emerges as the budget permits purchases. With an action limit of 12, for example, SR scores 28.4, compared with 19.7 for Full and 24.5 for Mask-Only. Together with the local behavior and rework analyses, this trend supports passing necessary guidance through subgoals while constraining local actions, allowing limited interactions to contribute more effectively to task progress.

Time budgets further reveal the cost differences between handoff strategies. TH outperforms Full under action budgets, but generating additional instructions consumes time that could otherwise be used for execution. Under the 30-second WebShop budget, TH scores 21.1, below Full's 24.5, while SR reaches 32.5. TH's task gains become more apparent as the budget increases. By completing the handoff within internal representations, SR retains guidance while avoiding instruction generation, leaving more time for task progress.

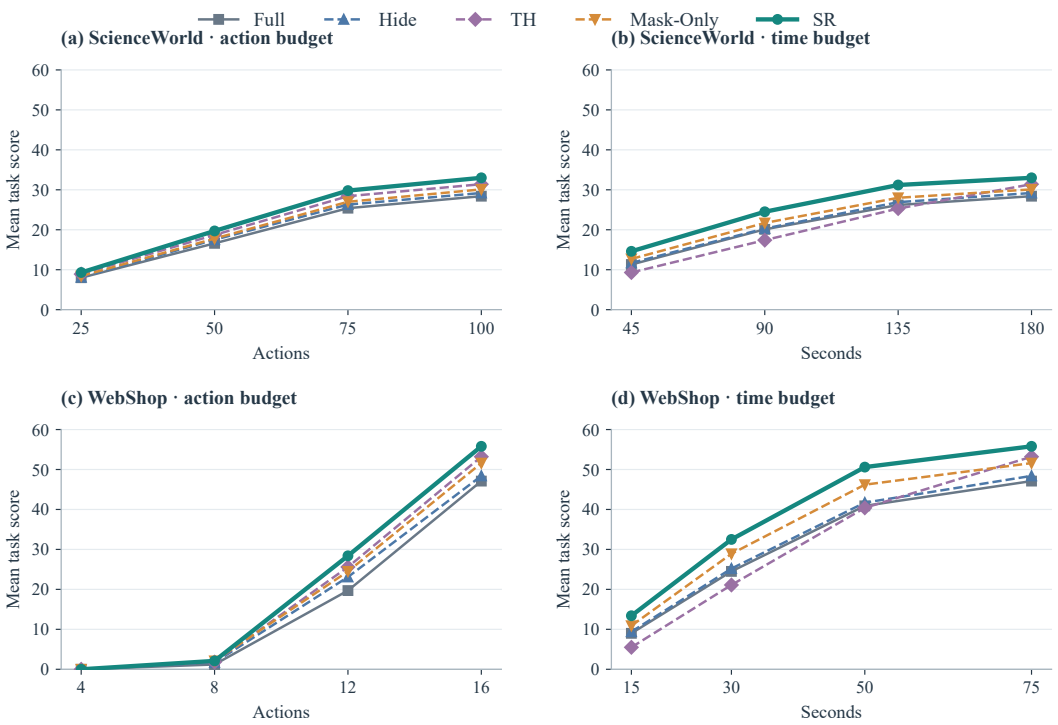

Figure D.3: Task scores under matched budgets for ReCAP with Qwen3-8B, comparing Full, Hide, TH, Mask-Only, and SR. Higher is better. (a,b) Mean final score on ScienceWorld. (c,d) Mean purchase reward multiplied by 100 on WebShop, with zero for no purchase. Action sweeps fix time limits at 180/75 seconds, and time sweeps fix action limits at 100/16 for ScienceWorld/WebShop. Curves truncate the same records at action or time boundaries and count rewards obtained before the cutoff. Maximum-budget points match Table 2. Time includes TH instruction generation, SR computation, all online processing, and environment interaction.

E Behavior Diagnostics

E.1 Main-Text Diagnostic Definitions

Local diagnostics for ReCAP with Qwen3-8B on ScienceWorld. All rates are percentages. Panels (a,b) show boundary violations and invalid decisions on the first call from matched entry states, both lower is better. Boundary violations include violated preconditions, actions outside the subgoal scope, or incorrect termination. Invalid decisions are empty responses or parsing failures. Panel (c) measures goal-requirement satisfaction: local behavior respects final-goal requirements relevant to the current subtask. These requirements appear only in the final-goal field of the selected inputs. Panel (d) measures whether subsequent execution completes the full remaining task. Higher is better in both panels. The binary scoring unit, evaluation window, and return-of-control criteria are defined in Appendix E.3. The two groups contain 96/64 source tasks and 384/256 entry states, respectively, with 5 repetitions. Results are aggregated within tasks and then averaged with equal task weights. Intervals are omitted.

E.2 Missing Information and Erroneous or Conflicting Subgoals

We compare subsequent execution from two types of diagnostic entry states: missing information and errors or conflicts. All conditions continue from the same states and retain ReCAP's original feedback-based revision mechanism. Appropriate handling includes grounded information gathering, correct return of control, or progress after planner revision. Remaining-task success is the fraction of continuations that complete the full task.

Table E.1: Continuation outcomes from entry states with missing information or errors/conflicts on ScienceWorld, using ReCAP with Qwen3-8B. All metrics are percentages, with higher values better. Appropriate handling means grounded information gathering, correct return of control, or progress after planner revision. Empty responses and mechanical execution of incorrect instructions do not qualify. Remaining-task success measures full-task completion after the entry state, not the main-table Score. Entry states are aggregated within source tasks and then averaged with equal task weights, separately for the two groups. Results are means $\pm$ sample standard deviations over five seeds. Each group contains 64 source tasks and 256 entry states. Strata and criteria are fixed before inspecting outputs.

Condition	Missing information		Errors/conflicts	
	Appropriate handling	Remaining-task success	Appropriate handling	Remaining-task success
Full	68.4 ± 3.1	31.6 ± 2.7	62.7 ± 3.6	23.8 ± 2.9
Hide	49.8 ± 3.8	21.2 ± 2.6	40.8 ± 4.1	15.7 ± 2.4
Mask-Only	54.2 ± 3.5	25.4 ± 2.8	45.6 ± 3.9	17.9 ± 2.6
SR	73.6 ± 2.9	39.8 ± 3.0	60.9 ± 3.4	27.4 ± 3.1

At entry states with missing information, Hide and Mask-Only have lower appropriate-handling and remaining-task success rates than Full. This exposes a cost of restricting goal access when the subgoal does not fully express the task's requirements. SR achieves 73.6% appropriate handling and 39.8% remaining-task success, above Full's 68.4% and 31.6%. Here, retaining guidance supports grounded information gathering and subsequent progress, extending the diagnostic beyond execution with complete subgoal information.

Entry states with errors or conflicts show a different relationship between the metrics. Full achieves 62.7% appropriate handling, slightly above SR's 60.9%, but SR achieves 27.4% remaining-task success, above Full's 23.8%. Even after a local issue is handled appropriately, subsequent execution determines whether the task is completed, so the rankings need not coincide. With the original feedback-based revision mechanism retained, these results show that SR's local execution constraints can coexist with information gathering, return of control, and subsequent task progress.

E.3 Statistics and Joint Analysis of Behavior Diagnostics

The main-experiment local diagnostics use preselected shared entry states, with the same context and environment state across conditions. Each source task provides four entry states, and each entry state is repeated five times. The motivation comparisons instead use one entry per task and one execution per condition (Appendix C.2). Results are first aggregated within tasks and then averaged with equal source-task weights. Group sizes are provided in the corresponding figures and tables. Diagnostic subsets may overlap and are not summed into a total of independent tasks.

Continuation budgets. For the ScienceWorld diagnostics, each restored entry receives a fresh budget of 100 environment operations and 180 seconds, shared across conditions. Timing starts after restoration and covers model computation, controller processing, and environment interaction. The local evaluation window ends at subgoal completion, return of control, or execution stop. Subsequent execution continues under the same budget, without resetting it when the local window ends.

Behavior criteria and label assignment. Task and subgoal completion are determined by environment evaluators. Empty responses and parsing failures are identified by programmatic rules. A boundary violation occurs when, before completing the current subgoal, the executor advances to another stage without authorization, violates the current execution preconditions, or incorrectly terminates the current work. Necessary preparation, appropriate return of control, and authorized plan revisions are excluded.

For behaviors requiring semantic judgment, we use GPT-5 to assign initial labels under a common set of criteria. The assessment input contains the final goal, current subgoal, relevant environment information, and execution records. A human reviewer then checks each label and corrects errors. All reported behavior statistics use the reviewed labels, with the same criteria applied across conditions.

Behavior rates are first aggregated within each source task and then averaged with equal task weights, so tasks with more diagnostic segments do not receive greater weight.

Goal-requirement satisfaction. Each local execution from a matched entry receives one binary label per condition and repetition. The evaluation window ends when the current subgoal is completed, control returns to the higher level, or local execution stops, including at a budget limit. Requirements are identified at the entry and include those in the final goal that apply to the current subtask. A positive label requires either valid progress while respecting these requirements, or correct return of control after accurately identifying missing necessary conditions without violating applicable constraints. Violations, unjustified returns, empty responses, and stagnation receive negative labels. Correct return can satisfy this criterion but does not count as subgoal completion. The two positive outcomes, valid progress and justified return, are mutually exclusive in the final classification. When both occur, a fixed terminal-category rule assigns one outcome, while event-level labels are retained separately. All conditions use the same rules. Labels are averaged within each source task and then across tasks with equal weights.

Independent annotation agreement. In a separate reliability check, we sample 160 behavior segments per label category after deduplication, stratified by condition. Two annotators, blinded to condition names, label these segments independently, and disagreements are then adjudicated. Pilot annotation clarifies the distinctions among normal exploration, unsupported boundary violations, and appropriate return of control before the rules are frozen. Table E.2 reports agreement before adjudication, with agreement rates of 90.6%–93.8% and Cohen's κ of 0.804–0.841. These values measure agreement between the two human annotators on the sampled segments. The diagnostic rates use the full set of human-reviewed labels and the task-weighted aggregation described above.

Table E.2: Inter-annotator agreement computed before adjudication, with $n = 160$ in each row. The four counts are both positive, first annotator positive only, second annotator positive only, and both negative. We report agreement rate and Cohen's κ.

Label	Four-cell counts	Agreement (%)	κ
Boundary violations	49/7/5/99	92.5	0.834
Goal requirements	112/6/4/38	93.8	0.841
Appropriate handling	89/8/7/56	90.6	0.804

Table E.3: Joint counts of appropriate handling and remaining-task success from entry states with errors/conflicts. Each row contains 64 source tasks $\times$ 4 entry states $\times$ 5 repetitions, totaling 1,280 continuations. These continuations are not treated as independent tasks for inference. Marginal proportions recover Table E.1, while the joint distribution describes progress after handling the issue.

	Appropriate handling		Inappropriate handling	
Condition	Success	Failure	Success	Failure
Full	234	568	70	408
Hide	114	408	87	671
Mask-Only	136	448	93	603
SR	262	517	89	412

Table E.3 separates local handling from subsequent completion. Full has a slightly higher appropriate-handling rate, but the share of continuations with both appropriate handling and subsequent completion is 18.3%, below SR's 20.5%. Meanwhile, the share completed without appropriate handling increases from 5.5% to 7.0%. SR's overall completion gain is therefore not confined to continuations judged to handle the issue appropriately. This distribution shows that local behavior judgments and task outcomes capture different aspects of execution. Joint statistics provide information omitted when the two marginal rates are reported separately.

E.4 Behavior across Frameworks and Environments

We extend the local behavior analysis to HiAgent/ScienceWorld and ReCAP/ALFWorld, both with Qwen3-8B, comparing Full, Mask-Only, and SR. The first changes the base framework and the second changes the task environment. Both retain the main-experiment parameters and separately examine boundary constraints and necessary guidance. The boundary group uses entry states where currently visible information is sufficient for action. The necessary-guidance group uses naturally occurring entry states with requirements stated only in the overall-goal field. In HiAgent, interventions occur only at action entry states after the subgoal has been determined and the fields are valid. The original subgoal selection and switching procedures are retained.

Table E.4: Local behavior in two additional settings. All metrics are percentages, reported as means $\pm$ sample standard deviations over five repetitions. The first two columns belong to the boundary group, and the last two to the necessary-guidance group, with separate denominators. Each HiAgent group contains 64 source tasks and 256 entry states. Each ALFWorld group contains 48 source tasks and 192 entry states. Criteria follow Figure 3, and sampling and aggregation follow Appendix E.3. Lower boundary-violation and invalid-decision rates are better. Higher goal-requirement satisfaction and remaining-task success are better. Remaining-task success measures full-task completion after the diagnostic entry state, whereas the main ScienceWorld Score is mean final score. SR$-$Mask-Only goal-requirement-satisfaction differences and 95% intervals are 9.5 [4.5, 14.8] and 11.1 [5.5, 17.5] percentage points, respectively. Remaining-task success differences are 4.9 [-1.6, 11.4] and 6.6 [0.3, 13.0] percentage points. Differences and means are rounded independently, so subtraction of displayed values may differ by 0.1. Intervals follow Appendix D.11.

Condition	Boundary violations ↓	Invalid decisions ↓	Goal-requirement satisfaction ↑	Remaining-task success ↑
		HiAgent / ScienceWorld		
Full	21.2 ± 2.0	11.6 ± 1.6	88.5 ± 2.5	41.6 ± 2.8
Mask-Only	16.4 ± 2.1	15.2 ± 1.6	73.7 ± 2.5	40.3 ± 2.7
SR	13.6 ± 2.1	12.0 ± 1.7	83.1 ± 2.4	45.2 ± 2.7
		ReCAP / ALFWorld		
Full	26.2 ± 2.3	10.9 ± 1.7	90.4 ± 2.4	62.7 ± 2.6
Mask-Only	18.0 ± 2.1	14.6 ± 1.5	76.1 ± 2.5	61.8 ± 2.7
SR	13.1 ± 2.0	10.0 ± 1.6	87.3 ± 2.6	68.3 ± 2.7

The two settings show the same component relationship. Relative to Full, Mask-Only reduces boundary violations but increases invalid decisions and lowers goal-requirement satisfaction. Adding value transfer, SR further reduces boundary violations and improves both decision validity and goal-requirement satisfaction. These changes support the complementary roles of masking and transfer: the former restricts the overall goal's direct influence on local actions, while the latter restores guidance impaired by masking alone. The same relationship persists in HiAgent despite its existing subgoal-based memory management, supporting the complementary value of history management and changes to goal guidance.

Relative to Mask-Only, SR's goal-requirement-satisfaction gain has a positive interval in both settings, although satisfaction does not fully recover to Full's level. Subsequent completion differs across settings. The ReCAP/ALFWorld gain has an interval above zero, whereas the HiAgent/ScienceWorld mean improves but its interval crosses zero. Both comparisons thus support partial recovery of necessary guidance, with clearer support on ALFWorld for this improvement extending to full-task completion.