FROM TRAJECTORY EVIDENCE TO CONTROL: LATENT CONTROL-STATE INFERENCE FOR LONG-HORIZON LANGUAGE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language agents often fail in long-horizon tasks despite making locally reasonable decisions. A recurring difficulty is that accumulated trajectory evidence may call for a change in how the agent should proceed, while next-action generation continues along the current course. We formulate this missing intermediate decision as *latent control-state inference* and introduce an Explore–Exploit–Recover (E^2R) controller. At each step, E^2R infers a trajectory-conditioned control state consisting of a control mode and a trajectory-specific target, and uses it to regulate the proposed action: EXPLORE resolves missing task-relevant evidence, EXPLOIT advances a supported course of action, and RECOVER breaks from one rendered ineffective by feedback. We evaluate E^2R on ALFWorld, WebArena-Lite, and ToolSandbox. Across environments, E^2R consistently improves end-to-end task performance and performance accumulated over the interaction budget. Behavioral analysis shows that E^2R acts as a sparse, mode-conditioned control layer, preserving most proposals while concentrating explicit corrections under particular inferred control states. Controlled interventions further indicate that both control-state content and its trajectory-aligned enactment contribute to the observed gains. These results suggest that effective long-horizon agency benefits from an explicit control layer between trajectory understanding and next-action generation.

1 INTRODUCTION

Language agents increasingly rely on natural-language reasoning to act over long interaction horizons (28; 3; 18; 15; 29). They plan and ground action sequences (21; 9; 6), reflect on failures and accumulated experience (19; 14; 30), and revise their behavior as interaction unfolds (25; 10).

Yet long-horizon interaction poses a problem beyond deciding *what action to take next*: accumulated trajectory evidence can change *how* the agent should proceed. When task-relevant evidence is missing, continued commitment may be premature; when the current course is well supported, further deliberation may be unnecessary; and when feedback renders a continuation ineffective, persisting with it can waste the remaining interaction budget. These situations call for qualitatively different control demands—information acquisition, committed execution, and course correction. Existing agent frameworks typically realize such adaptation through next-action generation, plan revision, reflection, or search over alternative action trajectories (28; 19; 23; 2; 31). In these formulations, however, the manner of continuation is generally embedded within the task-level decision process or agent architecture rather than represented as a separate trajectory-conditioned control variable.

In these formulations, however, the manner of continuation is generally embedded within the task-level decision process. Figure 1 illustrates one concrete manifestation of this gap: even after receiving explicit negative feedback, a ReAct agent can continue proposing the same failed action.

We formulate this missing intermediate decision as *latent control-state inference*. Rather than leaving adaptation implicit in next-action generation, we introduce a compact trajectory-conditioned control state consisting of a control mode $m_t \in \{\text{EXPLORE}, \text{EXPLOIT}, \text{RECOVER}\}$ and a trajectory-specific target ℓ_t. The mode specifies the type of control required, while the target identifies the trajectory-specific issue that the control should act on. EXPLORE prioritizes resolving missing

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.

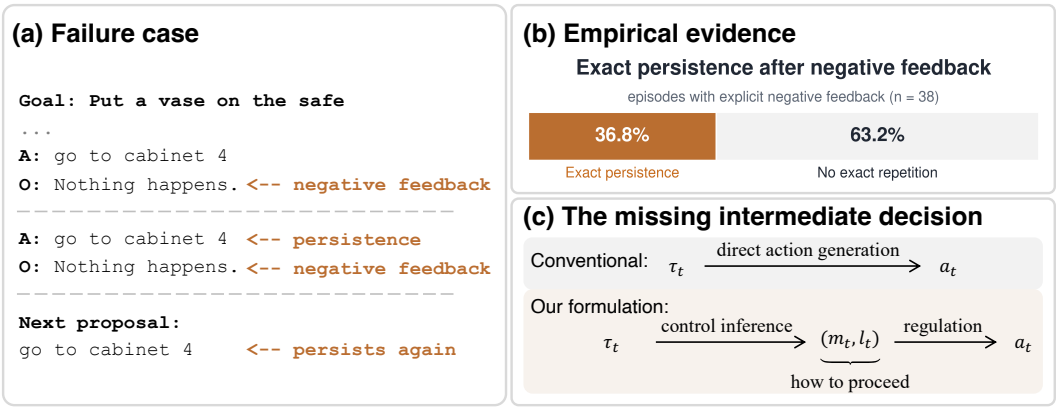


Figure 1: **Motivation for latent control-state inference.** **(a)** In an ALFWorld ReAct trajectory, the agent repeats the same executable action despite explicit negative feedback (`Nothing happens.`) and subsequently proposes it again. **(b)** Among 38 ReAct episodes containing at least one such negative-feedback event, 14 (36.8%) exhibit at least one *exact persistence* event, in which the next executable action exactly repeats the failed action. **(c)** Conventional next-action generation leaves the manner of continuation implicit. Our formulation introduces an intermediate trajectory-conditioned control state (m_t, ℓ_t) that represents *how to proceed* and regulates the base proposal before execution.

task-relevant evidence, EXPLOIT advances a supported commitment, and RECOVER breaks from a continuation rendered ineffective by feedback. Together, mode and target provide an explicit intermediate representation between trajectory evidence and task-level action.

Based on this formulation, we introduce E^2R, an Explore–Exploit–Recover controller that separates task-level action proposal from trajectory-conditioned regulation. At each step, the base agent produces a candidate action from the current trajectory. Given the trajectory and proposal, E^2R infers a control mode and target and determines whether the proposal is consistent with the resulting control state. Compatible proposals are preserved, whereas incompatible proposals are redirected through mode-specific regulation before execution. This decomposition leaves task-specific action generation to the base agent while making changes in control explicit as trajectory evidence accumulates. The same logical decomposition admits both a fused one-call realization for end-to-end evaluation and a decoupled realization that exposes proposal and control separately for mechanism studies.

We evaluate E^2R on ALFWorld (20), WebArena-Lite (11; 32), and ToolSandbox (13). Across all three environments, E^2R improves terminal task performance and performance–budget AUC. Trajectory-level analysis shows that the controller operates selectively: most base proposals are preserved, while redirects are concentrated under particular inferred control states, especially RECOVER. Matched control experiments further distinguish structured control from additional trajectory reasoning alone. The inferred control state provides larger gains than generic reconsideration, explicit mode-specific enactment improves over presenting the same control information only as feedback, and perturbing the enacted control state degrades performance. Together, these results indicate that the gains depend on both the trajectory-conditioned control decision and how that decision is translated into action.

Overall, our contributions are:

- We formulate long-horizon adaptation as *latent control-state inference*, introducing a trajectory-conditioned representation of both the required control mode and its trajectory-specific target.
- We introduce E^2R, a selective Explore–Exploit–Recover controller that separates task-level action proposal from trajectory-conditioned regulation, preserving compatible proposals while redirecting incompatible ones according to the inferred control state.

- Across three interactive benchmarks, we show that explicit control-state inference improves long-horizon performance, and use trajectory-level analysis and matched interventions to characterize when the controller acts and which components of structured control account for the gains.

2 Related Work

2.1 Reasoning and Adaptation in Language Agents

Large language models have increasingly been used as interactive agents by interleaving natural-language reasoning with environment actions. ReAct (28) established a widely used paradigm in which reasoning traces and actions are generated iteratively, allowing environment observations to inform subsequent decisions. Subsequent work has introduced richer mechanisms for long-horizon behavior, including verbal reflection and memory (19), explicit planning and replanning (4; 22; 23; 12), modular reasoning and tool execution (27), and search over alternative reasoning–action trajectories (31).

Our work focuses on a complementary aspect of long-horizon adaptation. In many language-agent architectures, trajectory feedback is incorporated within the same process that generates the next task-level decision. We instead introduce an explicit intermediate control abstraction between trajectory understanding and action generation. The abstraction specifies whether the agent should acquire missing evidence, advance a supported commitment, or break from an ineffective continuation, together with the trajectory-specific target of that control decision.

2.2 Feedback-Driven Self-Correction and Regulation

A closely related line of work studies how language models use self-generated or external feedback to revise their outputs and behavior. Self-Refine (14) iteratively critiques and revises an initial response, while Reflexion (19) converts task feedback into verbal reflections that guide later decisions. CRITIC (1) grounds correction in external tools that validate an initial response before revision. In interactive settings, feedback can support replanning and adaptation, as in Inner Monologue (4), while broader work has examined when feedback-driven self-correction is effective (7).

E^2R differs in how feedback is operationalized. Rather than using trajectory feedback only as additional context for another round of reasoning or generation, E^2R converts accumulated evidence into an explicit control signal before execution. It then evaluates the agent's proposal under that signal and selectively intervenes when the proposed action conflicts with the inferred control state. This creates a separation between *control inference* and *action regulation*: the former determines the trajectory-conditioned mode and target, while the latter determines how that signal should affect the concrete action. Our approach therefore complements feedback-driven self-correction by making the control consequence of trajectory evidence explicit.

2.3 Hierarchical and Latent Control Abstractions

Separating high-level control from low-level action selection has a long history in sequential decision making. SwiftSage switches between fast and deliberative agent modules (10), while System-1.x learns to allocate subgoals between fast and slow planning regimes (17). These approaches use high-level control primarily to select the computational or planning mechanism. E^2R instead represents the agent's current behavioral relation to trajectory evidence: whether to acquire missing evidence, advance a supported continuation, or leave one invalidated by feedback. Hierarchical reinforcement learning introduces abstractions above primitive actions, most notably through the options framework (24), where a higher-level policy selects among temporally extended behaviors. E^2R similarly introduces a control variable above the level of primitive actions, but its modes are not task-specific skills or temporally extended options. Explore, Exploit, and Recover instead describe how an otherwise unchanged actor should respond to the current trajectory.

Our formulation is also related to latent-context inference in meta-reinforcement learning. Methods such as PEARL (16) infer latent task variables from interaction experience and condition behavior on the resulting representation, while VariBAD (33) performs approximate inference over latent task

uncertainty to support adaptive control. Related approaches similarly infer hidden task or environment context from experience (5). In contrast, our latent variable summarizes the agent's current *control relation to the trajectory*: whether progress calls for further evidence acquisition, continued commitment, or course correction. This yields a task-agnostic meta-control abstraction that can be applied on top of pretrained language agents without learning a new task policy or a library of task-specific skills.

3 METHOD

We formulate long-horizon adaptation as *latent control-state inference* and instantiate it with E^2R, a trajectory-conditioned controller that separates task-level action proposal from trajectory-level regulation. Figure 2 provides an overview: the controller infers a compact control state from accumulated trajectory evidence, which is then used to selectively selectively regulate the action proposal.

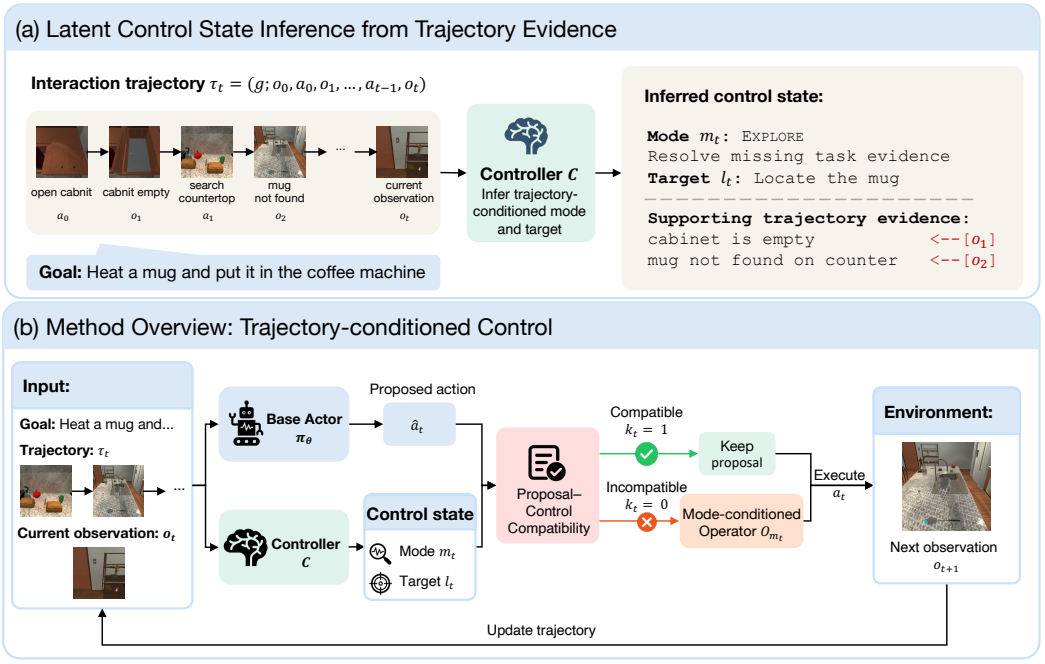


Figure 2: **Overview of E^2R. (a)** From accumulated trajectory evidence, the controller infers a control state $z_t = (m_t, \ell_t)$ specifying both the required control mode and its trajectory-specific target. **(b)** The base actor independently proposes $\hat{a}_t$. Compatible proposals are executed unchanged, while incompatible proposals are redirected by a mode-conditioned operator. The resulting observation updates the trajectory, closing the control loop.

3.1 LATENT CONTROL-STATE INFERENCE

Consider a language agent interacting with an environment over multiple steps. At step t, let

$$\tau_t = (g; o_0, a_0, o_1, \ldots, a_{t-1}, o_t) \tag{1}$$

denote the interaction trajectory, where g is the task goal, o_t the current observation, and a_t an environment action. A conventional language agent directly maps the accumulated trajectory to the next action,

$$a_t \sim \pi_\theta(a \mid \tau_t). \tag{2}$$

Long-horizon interaction introduces an additional decision between trajectory understanding and concrete action selection. Accumulated evidence may indicate that the agent should seek further information, continue a productive course, or break from a continuation that has become ineffective.

We represent this intermediate decision with a trajectory-conditioned control state

$$z_t = (m_t, \ell_t) = C(\tau_t), \tag{3}$$

where $m_t \in \mathcal{M} = \{\text{EXPLORE}, \text{EXPLOIT}, \text{RECOVER}\}$ is a low-cardinality control mode and ℓ_t is a trajectory-specific control target.

The mode m_t specifies the type of control required, while ℓ_t grounds that control in a trajectory-specific objective. Their operational semantics are:

$$m_t = \begin{cases} \text{EXPLORE}, & \text{resolve task-relevant evidence needed for further progress,} \\ \text{EXPLOIT}, & \text{advance a commitment currently supported by the trajectory,} \\ \text{RECOVER}, & \text{break from a continuation rendered ineffective by feedback.} \end{cases} \tag{4}$$

Accordingly, ℓ_t identifies the missing evidence to resolve under EXPLORE, the supported commitment to advance under EXPLOIT, and the ineffective continuation to leave under RECOVER.

The control state depends on the accumulated trajectory rather than on the current observation alone. The same local observation may therefore induce different control states depending on what the agent has already established, attempted, or received as feedback.

3.2 E^2R: Separating Action Proposal from Trajectory-Level Control

Given τ_t, the base agent proposes

$$\hat{a}_t \sim \pi_\theta(a \mid \tau_t), \tag{5}$$

while, in parallel, the controller constructs

$$(m_t, \ell_t) = C(\tau_t). \tag{6}$$

The proposal and inferred control state are then passed to a selective regulator, detailed below. This decomposition leaves task-specific action selection to the base agent, while the controller determines how accumulated trajectory evidence should regulate that proposal.

3.3 Mode-Conditioned Action Regulation

A proposal need not realize the control requirement implied by the trajectory. Given $\hat{a}_t$ and (m_t, ℓ_t), a compatibility function determines whether the proposal is consistent with the inferred control state:

$$k_t = K(\hat{a}_t; m_t, \ell_t, \tau_t) \in \{0, 1\}. \tag{7}$$

Under EXPLORE, a compatible proposal acquires information relevant to ℓ_t; under EXPLOIT, it advances the commitment represented by ℓ_t; and under RECOVER, it avoids continued reliance on the ineffective continuation identified by ℓ_t.

The regulator follows the selective intervention rule

$$a_t = \begin{cases} \hat{a}_t, & k_t = 1, \\ O_{m_t}(\tau_t, \ell_t, \hat{a}_t), & k_t = 0, \end{cases} \tag{8}$$

where O_{m_t} is the operator associated with the inferred control mode. Thus, compatible proposals are preserved, while incompatible proposals are redirected through mode-conditioned regulation.

The operators instantiate the corresponding control semantics: O_{EXPLORE} redirects behavior toward resolving the missing evidence identified by ℓ_t; O_{EXPLOIT} advances the supported commitment without reopening resolved decisions; and O_{RECOVER} breaks from the ineffective continuation and redirects behavior toward an alternative course.

3.4 Closed-Loop Control over Interaction Trajectories

After executing a_t, the environment returns o_{t+1} and updates the trajectory as $\tau_{t+1} = (\tau_t, a_t, o_{t+1})$. The controller then reconstructs $(m_{t+1}, \ell_{t+1}) = C(\tau_{t+1})$.

Control therefore evolves with interaction. Acquiring missing evidence can move behavior from EXPLORE to EXPLOIT; negative feedback can move it from EXPLOIT to RECOVER; and recovery

can expose new uncertainty that again calls for exploration. These transitions are not scheduled in advance, but emerge as the control state is reconstructed from newly accumulated evidence.

Overall, E^2R implements

$$\tau_t \xrightarrow{C} (m_t, \ell_t), \qquad (\tau_t, \hat{a}_t, m_t, \ell_t) \xrightarrow{R} a_t, \tag{9}$$

placing a trajectory-conditioned control layer between proposal and execution. Implementation details, including controller prompts, compatibility criteria, mode-specific regulation instructions, and benchmark-specific action interfaces, are provided in Appendix A.

4 EXPERIMENTS

In this section, we evaluate E^2R across three interactive environments and address three questions: (1) does explicit trajectory-level control improve end-to-end task performance; (2) how does E^2R regulate the agent along the trajectory; and (3) do the gains depend on the inferred control state and its explicit enactment? We first describe the common experimental setup and then address each question in turn.

4.1 EXPERIMENTAL SETUP

Environments. We evaluate on three interactive environments spanning distinct forms of long-horizon decision making. **ALFWorld** (20) requires multi-step household interaction through textual observations and actions. **WebArena-Lite** (11), constructed from WebArena (32), evaluates browser-based navigation, information retrieval, and interaction with stateful websites. **ToolSandbox** (13) evaluates structured tool use through stateful function calls and provides graded evaluation of intermediate and final task states.

Baselines. We compare E^2R with three representative agent architectures under the same backbone model and environment interface. **Vanilla** directly predicts the next action from the task goal and interaction history. **ReAct** (28) interleaves language reasoning and environment actions. **Plan-and-Execute** separates high-level task planning from execution, with replanning when necessary. We additionally evaluate **Reflexion** (19) in its multi-trial setting with cross-trial reflective memory. Within each environment, all methods use DeepSeek V4.1-flash (26) as the language backend and share the same configuration and environment interface. Episodes terminate upon task completion, environment termination, or budget exhaustion, and begin from a fresh agent state unless otherwise specified.

Evaluation metrics. We report terminal task performance together with environment-interaction efficiency. For ALFWorld and WebArena-Lite, terminal performance is success rate,

$$\mathrm{SR} = \frac{1}{N}\sum_{i=1}^{N} y_i, \tag{10}$$

where $y_i \in \{0, 1\}$ indicates whether episode i succeeds. For ToolSandbox, we report its native graded score,

$$\mathrm{Score} = \frac{1}{N}\sum_{i=1}^{N} s_i, \tag{11}$$

where s_i is the terminal score of episode i.

We additionally report Performance–Budget AUC (**PB-AUC**) to capture how performance evolves with the environment-interaction budget. Let c_i denote the budget consumed by episode i, and define its budget-dependent performance as

$$p_i(b) = v_i \mathbb{I}[c_i \le b], \qquad v_i = \begin{cases} y_i, & \text{for ALFWorld and WebArena-Lite,} \\ s_i, & \text{for ToolSandbox.} \end{cases} \tag{12}$$

Here, c_i is measured in environment steps for ALFWorld and WebArena-Lite, and in the benchmark's effective turns for ToolSandbox. The aggregate performance at budget b is $P(b) =$

$\frac{1}{N}\sum_{i=1}^{N} p_i(b)$. For an environment-specific maximum budget B, we compute

$$\text{PB-AUC} = \frac{1}{B}\sum_{b=1}^{B}\frac{P(b-1)+P(b)}{2}. \tag{13}$$

Higher PB-AUC indicates that an agent obtains stronger terminal performance using a smaller fraction of the available budget. We report SR and PB-AUC for ALFWorld and WebArena-Lite, and Score and PB-AUC for ToolSandbox.

4.2 RQ1: Does Explicit Control Improve Long-Horizon Task Performance?

We first evaluate whether explicit trajectory-level control improves end-to-end agent performance. Table 1 compares E^2R with Vanilla, ReAct, and Plan&Execute under the same backbone model, environment interface, and inference budget. We report both terminal task performance and PB-AUC, which captures performance accumulated over the environment-interaction budget.

Table 1: End-to-end performance across interactive environments. SR denotes task success rate, Score denotes the native graded task score in ToolSandbox, and PB-AUC measures performance accumulated over the interaction budget. Higher is better for all metrics.

Method	ALFWorld		WebArena-Lite		ToolSandbox	
	SR (%)	PB-AUC	SR (%)	PB-AUC	Score	PB-AUC
Vanilla	47.76	0.3125	23.03	0.1999	0.8136	0.5526
ReAct	70.90	0.5369	24.24	0.2079	0.8033	0.5469
Plan&Execute	50.00	0.3693	21.21	0.1977	0.7710	0.4953
E^2R	88.81	0.6538	28.48	0.2346	0.8338	0.5872

E^2R achieves the best terminal performance and PB-AUC in all three environments. On ALFWorld, it improves SR from 70.90% with ReAct to 88.81%, while increasing PB-AUC from 0.5369 to 0.6538. On WebArena-Lite, SR increases from 24.24% to 28.48% and PB-AUC from 0.2079 to 0.2346. On ToolSandbox, E^2R obtains a score of 0.8338 and PB-AUC of 0.5872, compared with $0.8136/0.5526$ for the strongest competing results across the two metrics. These results show consistent gains across distinct interaction settings and under both binary-success and graded evaluation.

Complementarity with episodic reflection. Reflexion operates across repeated task attempts, whereas our main comparison considers single-run agents. We therefore evaluate Reflexion separately on ALFWorld. As shown in Table 2, increasing the number of trials raises eventual success, with Reflexion-10 reaching 100% SR. However, single-trajectory E^2R attains a higher PB-AUC than Reflexion-10 (0.6538 vs. 0.6300). The two approaches are also complementary: adding E^2R raises Reflexion-3 from 75.40% to 92.64% SR and from 0.4319 to 0.6913 PB-AUC; with ten trials, both reach 100% SR, while PB-AUC increases from 0.6300 to 0.7901. Overall, E^2R consistently improves both terminal task performance and performance accumulated over the interaction budget. We next examine how this control is expressed along the trajectory.

Table 2: Comparison with multi-trial Reflexion on ALFWorld. Reflexion-K allows up to K task attempts with cross-trial reflective memory. PB-AUC accounts for the interaction budget consumed across attempts.

Method	SR (%)	PB-AUC
ReAct	70.90	0.5369
E^2R	88.81	0.6538
Reflexion-3	75.40	0.4319
Reflexion-3 + E^2R	92.64	0.6913
Reflexion-10	100.00	0.6300
Reflexion-10 + E^2R	100.00	0.7901

4.3 RQ2: How Does E^2R Regulate the Action?

RQ1 shows that E^2R improves end-to-end performance. We next examine how this control is expressed along the trajectory, focusing on mode allocation and explicit modification of the proposal.

Control profile and intervention sparsity. Let $\hat{a}_t$ denote the agent's proposed action and a_t the action executed after control. We define a *redirect* as

$$r_{i,t} = \mathbb{I}\left[\text{norm}(a_{i,t}) \neq \text{norm}(\hat{a}_{i,t})\right]. \tag{14}$$

We report the fraction of steps assigned to each control mode, the fraction of redirected steps (*Redirect Rate*), the fraction of episodes containing at least one redirect (*Intervention Coverage*), and the average number of redirects per episode. Mode Usage and Redirect Rate are computed within each episode and then macro-averaged across episodes.

Table 3: Control profile and intervention behavior of E^2R across environments.

Environment	Explore	Exploit	Recover	Redirect Rate	Intervention Coverage	Redirects/Episode
ALFWorld	43.20%	46.60%	10.20%	1.00%	15.70%	0.31
WebArena-Lite	45.50%	33.20%	21.30%	0.40%	5.20%	0.06
ToolSandbox	19.10%	60.50%	20.40%	6.80%	23.60%	0.59

Table 3 shows distinct control profiles across environments. ALFWorld allocates similar proportions of steps to EXPLORE and EXPLOIT, ToolSandbox is dominated by EXPLOIT, while WebArena-Lite invokes RECOVER more frequently. Despite these differences, explicit action modification remains sparse, affecting only 0.4–6.8% of steps and 5.2–23.6% of episodes. Within trajectories that are redirected, intervention can also recur: redirected episodes contain 2.00 redirects on average in ALFWorld and 2.52 in ToolSandbox. Overall, the controller leaves most proposals unchanged while concentrating intervention within a subset of trajectories.

Redirects are strongly mode-conditioned. We next examine how intervention varies with the inferred control state by computing the step-pooled conditional redirect probability

$$P(\text{Redirect} \mid m) = \frac{\sum_{i=1}^{N}\sum_{t=1}^{T_i} \mathbb{I}[m_{i,t} = m]\, r_{i,t}}{\sum_{i=1}^{N}\sum_{t=1}^{T_i} \mathbb{I}[m_{i,t} = m]}. \tag{15}$$

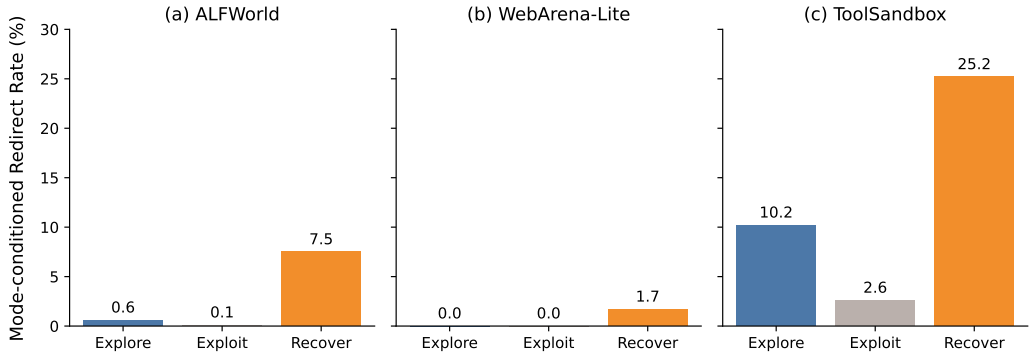


Figure 3: **Mode-conditioned intervention behavior of E^2R.** Probability of redirecting the proposal conditioned on the inferred control mode, computed by pooling eligible control steps across episodes.

Figure 3 shows a consistent mode-dependent intervention pattern. In ALFWorld, redirect rates are 0.6%, 0.1%, and 7.5% under EXPLORE, EXPLOIT, and RECOVER, respectively. In WebArena-Lite, all observed redirects occur under RECOVER, with a conditional redirect rate of 1.7%. ToolSandbox exhibits the same ordering, with redirect rates of 10.2%, 2.6%, and 25.2%.

Redirects are also disproportionately concentrated under RECOVER. Although RECOVER accounts for only 10.2%, 21.3%, and 20.4% of control steps in ALFWorld, WebArena-Lite, and ToolSandbox, respectively, it contributes 83.3%, 100.0%, and 64.2% of all redirects.

Together, these results reveal a selective, mode-conditioned control pattern: E^2R retains most proposals, while explicit intervention is concentrated under specific control states, particularly RECOVER. RQ3 next examines whether this structured control contributes to the observed performance gains.

4.4 RQ3: Are the Gains Due to Structured Control?

RQ1 establishes the end-to-end gains of E^2R, while RQ2 characterizes its control behavior. We next examine whether these gains arise from additional trajectory reasoning, structured control information, or explicit enactment. We use the decoupled realization of E^2R, where every condition starts from a proposal generated by the same standalone ReAct policy. All variants are evaluated on the full 134-task ALFWorld set with the base policy and interaction protocol fixed.

Matched controls. We compare five variants of the control pipeline. **Base (ReAct)** executes the base proposal directly. **Matched Reasoning** adds trajectory-level assessment and proposal reconsideration without an explicit control state. **Mode Feedback** infers the same mode and trajectory-specific target as E^2R, but provides them only as textual guidance. **Perturbed Control** retains explicit intervention while replacing the inferred state with a mismatched mode and target. Finally, E^2R explicitly enacts the inferred control state through mode-specific regulation.

Table 4: Controlled interventions on the E^2R pipeline. Mode and Target specify the control state available to regulation, while Enactment describes how it affects the base proposal. Higher is better.

Method	Control Pipeline			Performance	
	Mode	Target	Enactment	SR (%) ↑	PB-AUC ↑
Base (ReAct)	–	–	–	70.9	0.537
Matched Reasoning	–	–	Generic	82.8	0.596
Mode Feedback	Inferred	Inferred	Implicit	83.6	0.630
Perturbed Control	Perturbed	Perturbed	Explicit	83.6	0.627
E^2R	Inferred	Inferred	Explicit	88.8	0.654

Additional reasoning and structured control. Matched Reasoning already provides a substantial improvement over ReAct, from 70.9% to 82.8% SR and from 0.537 to 0.596 PB-AUC. Thus, an additional trajectory-level reconsideration step explains part of the overall gain. Introducing the inferred mode and target through Mode Feedback yields a modest further increase in SR ($82.8\% \rightarrow 83.6\%$) and a larger change in PB-AUC ($0.596 \rightarrow 0.630$), indicating that the structured control state provides information beyond generic reconsideration.

Enactment and control-state alignment. Explicitly enacting the inferred state further improves performance to 88.8% SR and 0.654 PB-AUC, compared with 83.6% and 0.630 under Mode Feedback. This suggests that using the control state to regulate the proposal is more effective than exposing it only as textual guidance.

Perturbing the enacted state reduces performance to 83.6% SR and 0.627 PB-AUC despite retaining explicit intervention. The comparison with E^2R therefore shows that the benefit is not explained by intervention alone; the enacted regulation also needs to correspond to the control state inferred from the trajectory.

Overall, the results show that additional trajectory reasoning contributes substantially, while structured control provides further gains when the inferred state is explicitly and consistently enacted.

5 Conclusion

Long-horizon language agents must decide not only what action to take, but also when accumulated trajectory evidence calls for a change in how they proceed. We formulate this problem as *latent control-state inference* and introduce E^2R, an Explore–Exploit–Recover controller that infers a trajectory-conditioned control state to regulate a agent's proposed actions. Across three interactive environments, E^2R improves end-to-end performance and budget efficiency, while controlled analyses show that the gains arise from meaningful control-state inference and action regulation rather than additional reasoning alone. These results suggest that explicitly reasoning about *how to proceed* from trajectory evidence is a useful intermediate abstraction for building more adaptive long-horizon language agents.

AI USE STATEMENT

Generative AI tools were used to assist with drafting and polishing portions of the manuscript, including improving clarity, organization, and presentation. They were also used for literature retrieval and discovery, such as identifying potentially relevant prior work. All research decisions, experimental procedures, reported results, and AI-assisted content were reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

REFERENCES

[1] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024.

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

[3] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 9118–9147. PMLR, 2022.

[4] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 1769–1782. PMLR, 2023.

[5] Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A. Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

[6] Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T. Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR, 2023.

[7] Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024.

[8] Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Russ Salakhutdinov, and Nicolas Chapados. The browsergym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025.

[9] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[10] Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[11] Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Xixuan Song, Yifan Xu, Shudan Zhang, Hanyu Lai, Jiadai Sun, Xinyue Yang, Yu Yang, Zehan Qi, Shuntian Yao, Xueqiao Sun, Siyi Cheng, Qinkai Zheng, Hao Yu, Hanchen Zhang, Wenyi Hong, Ming Ding, Lihang Pan, Xiaotao Gu, Aohan Zeng, Zhengxiao Du, Chan Hee Song, Yu Su, Yuxiao Dong, and Jie Tang. Visualagentbench: Towards large multimodal models as visual foundation agents. In *International Conference on Learning Representations*, 2025.

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

[13] Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183. Association for Computational Linguistics, 2025.

[14] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[15] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22. Association for Computing Machinery, 2023.

[16] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5331–5340. PMLR, 2019.

[17] Swarnadeep Saha, Archiki Prasad, Justin Chih-Yao Chen, Peter Hase, Elias Stengel-Eskin, and Mohit Bansal. System 1.x: Learning to balance fast and slow planning with language models. In *International Conference on Learning Representations*, 2025.

[18] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

[19] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

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

[21] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[22] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. LLM-Planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009, 2023.

[23] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[24] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.

[25] 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 *Advances in Neural Information Processing Systems*, volume 36, pages 34153–34189, 2023.

[26] Anyi Xu, B Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, CC Yu, et al. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

[27] Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023.

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

[29] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077, Bangkok, Thailand, 2024. Association for Computational Linguistics.

[30] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

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

[32] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

[33] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: A very good method for bayes-adaptive deep RL via meta-learning. In *International Conference on Learning Representations*, 2020.

A Algorithm and Implementation Details

A.1 Logical Inference Procedure

E^2R is defined as a trajectory-level controller over a base language agent. At interaction step t, the base policy first forms a candidate environment action

$$\hat{a}_t \sim \pi_\theta(\cdot \mid \tau_t), \tag{16}$$

where τ_t denotes the complete trajectory history available at the current decision step. In our experiments, π_θ is instantiated by ReAct.

From the same trajectory, the controller constructs a trajectory-conditioned control state

$$(m_t, \ell_t) = C(\tau_t), \tag{17}$$

where $m_t \in \{\textsc{Explore}, \textsc{Exploit}, \textsc{Recover}\}$ is the control mode, e_t denotes trajectory evidence supporting the inferred state, and ℓ_t is a trajectory-specific control target.

The regulator then combines the base proposal with the inferred control state:

$$a_t = R(\tau_t, \hat{a}_t, m_t, e_t, \ell_t). \tag{18}$$

The regulator preserves $\hat{a}_t$ when the proposal already serves the inferred control objective and otherwise redirects it toward an action that better realizes that objective.

Algorithm 1 summarizes the logical procedure.

Algorithm 1 E^2R logical inference procedure

Require: base policy π_θ, controller C, regulator R, interaction budget T

Initialize trajectory τ_0 from the task and initial observation
for $t = 0, \ldots, T-1$ **do**
  $\hat{a}_t \sim \pi_\theta(\cdot \mid \tau_t)$
  $(m_t, \ell_t) \leftarrow C(\tau_t)$
  $a_t \leftarrow R(\tau_t, \hat{a}_t, m_t, e_t, \ell_t)$
  Execute a_t and receive observation o_{t+1}
  $\tau_{t+1} \leftarrow (\tau_t, a_t, o_{t+1})$
  if the environment terminates **then**
    break
  end if
end for

A.2 Control-State Semantics

The control mode specifies how the next action should respond to evidence in the current interaction trajectory. Its operational interpretation is summarized in Table 5.

Table 5: Operational interpretation of the three control modes.

Mode	Control objective
Explore	Acquire a specific missing task-relevant fact when the current continuation has not been invalidated.
Exploit	Directly advance a continuation already supported by trajectory evidence.
Recover	Repair, bypass, or leave a continuation rendered ineffective by observed failure, contradiction, violated preconditions, or lack of progress.

The three modes distinguish different relationships between accumulated trajectory evidence and immediate behavior. Recover takes precedence while an observed failure remains unresolved. A diagnostic action whose purpose is to repair or leave such a failure therefore remains part of Recover. Explore applies when the current continuation has not been invalidated but a specific

task-relevant fact remains unresolved. EXPLOIT applies when the trajectory already supports a productive continuation.

The control target ℓ_t instantiates the selected mode for the current trajectory. For EXPLORE, it identifies the missing information to resolve; for EXPLOIT, the supported continuation to advance; and for RECOVER, the failed continuation to repair, bypass, or leave. The evidence variable e_t records the trajectory evidence supporting this decision and is restricted to information already present in τ_t.

A.3 IMPLEMENTATIONS

Algorithm 1 specifies the logical decomposition of E^2R, but does not require each logical operation to correspond to a separate language-model invocation. We instantiate the same proposal–control–regulation structure in two forms: a *decoupled* realization that generates the base proposal independently, and a *fused* realization that serializes the operations within a single completion for inference efficiency.

Decoupled E^2R. The decoupled realization explicitly isolates proposal generation from the E^2R controller. At each environment step, we first invoke the standalone ReAct policy using the same task context, demonstrations, prompt template, and decoding configuration as the ReAct baseline:

$$\hat{a}_t \sim \pi_{\mathrm{ReAct}}(\cdot \mid \tau_t). \tag{19}$$

No E^2R instruction is present during proposal generation.

The resulting proposal $\hat{a}_t$ is then held fixed. A second language-model invocation receives the current trajectory together with this fixed proposal and realizes the remaining controller operations:

$$(m_t, e_t, \ell_t, a_t) = F_{\mathrm{ctrl}}(\tau_t, \hat{a}_t). \tag{20}$$

Within this invocation, the control state (m_t, e_t, ℓ_t) is inferred from the trajectory according to the semantics of $C(\tau_t)$. The proposal $\hat{a}_t$ is subsequently evaluated against this inferred control state. If the proposal already realizes the active control objective, it is preserved; otherwise, the model returns a mode-consistent replacement action.

Thus, the decoupled realization requires two language-model calls per environment step: one call for the standalone ReAct proposal and one call for control-state inference and regulation. Its key property is that the proposal is generated by exactly the standalone ReAct policy before any E^2R instruction is introduced.

Fused E^2R. For the main benchmark experiments, we use an inference-efficient fused realization. The proposal, control state, and final action are serialized within a single structured completion:

$$(\tilde{a}_t, m_t, e_t, \ell_t, a_t) = F_{\mathrm{fused}}(\tau_t), \tag{21}$$

where $\tilde{a}_t$ denotes the pre-regulation proposal produced inside the fused completion.

The output fields follow the logical order of Algorithm 1: the model first materializes a candidate action, then constructs the trajectory-conditioned control state, and finally returns the action executed after regulation. The evidence supporting (m_t, e_t, ℓ_t) is required to refer only to information contained in the current task trajectory.

The decoupled realization provides the corresponding controlled setting in which proposal generation is strictly separated from E^2R. The fused realization requires one language-model call per environment step and is used for the main end-to-end evaluation.

A.4 COMPARISON OF THE TWO REALIZATIONS

We compare the decoupled and fused realizations on ALFWorld using the same language model, task set, environment interface, and decoding configuration. We additionally report standalone ReAct to test whether the controller remains effective when proposal generation is completely separated from the E^2R prompt.

The decoupled realization improves substantially over standalone ReAct even when the proposal is generated independently by the exact ReAct policy. Its terminal performance matches the fused

Table 6: Comparison of standalone ReAct and the two realizations of E^2R on ALFWorld. The decoupled realization generates the proposal using the exact standalone ReAct policy before invoking the controller, whereas the fused realization serializes proposal formation and control within one completion.

Method	LM calls / step	SR (%)	PB-AUC
ReAct	1	70.90	0.5369
E^2R-Decoupled	2	88.81	0.6544
E^2R-Fused	1	88.81	0.6538

realization, with similar PB-AUC. This comparison shows that the benefit of E^2R persists when proposal generation is isolated from the control prompt, while the fused realization provides a more inference-efficient implementation for end-to-end evaluation.

B PROMPTS

This section provides the prompt templates used by the base agents, the E^2R controller, and the controlled variants in our mechanism study. Variables enclosed in braces are replaced at runtime by the corresponding task context, trajectory, proposal, or control state. Unless otherwise stated, no task-specific instructions are added beyond those supplied by the environment.

B.1 BASE AGENT PROMPTS

ReAct and the proposal policy of E^2R. E^2R uses ReAct as its underlying proposal policy. ReAct requires exactly one language-model call per environment step. Each call produces a single completion containing a short reasoning trace followed by one candidate executable action. The candidate action is parsed and executed by the environment, and the resulting observation is appended to the interaction history for the next turn.

For ALFWorld, we use two demonstrations from the same task family for each episode:

```
Interact with a household to solve a task. Here are two examples.
{TASK-FAMILY REACT DEMONSTRATION 1}
{TASK-FAMILY REACT DEMONSTRATION 2}

Here is the task.
{INITIAL OBSERVATION}
{TRAJECTORY}

>
```

Vanilla. The Vanilla agent directly generates the next executable action from the task specification and interaction trajectory, without an explicit reasoning trace or auxiliary control state:

```
{ENVIRONMENT AND TASK INSTRUCTIONS}

{INITIAL OBSERVATION}
{TRAJECTORY}

Output the next executable environment action.
```

Plan-and-Execute. Plan-and-Execute separates an explicit planning stage from subsequent action execution. The planner first constructs a task-level plan from the task specification and current trajectory:

```
{ENVIRONMENT AND TASK INSTRUCTIONS}
```

```
Task and interaction history:
{TRAJECTORY}

Construct a concise plan for completing the task.
Return the ordered plan steps only.
```

The executor is then conditioned on the current plan together with the latest interaction history:

```
{ENVIRONMENT AND TASK INSTRUCTIONS}

Current plan:
{PLAN}

Interaction history:
{TRAJECTORY}

Execute the next appropriate step of the plan.
Output exactly one executable environment action.
```

B.2 Decoupled Controller Prompt

In the decoupled realization, the first call is exactly the standalone ReAct prompt described in Section B.1. The resulting action proposal is held fixed and passed to the following controller prompt.

```
You are a trajectory-level controller for an interactive agent.

[Current task trajectory begins]
{CURRENT TASK TRAJECTORY}
[Current task trajectory ends]

The standalone ReAct agent proposed:
{PROPOSED ACTION}

First infer the active control state from the current task trajectory.

Control modes:

RECOVER:
Repair, bypass, or leave a continuation contradicted or rendered ineffective
by observed feedback.

EXPLORE:
Acquire a specific missing task-relevant fact when the current continuation
has not been invalidated.

EXPLOIT:
Advance a continuation already supported by trajectory evidence.

RECOVER takes precedence while an observed failure remains unresolved.

The control-state evidence must be grounded only in the marked task
trajectory. Do not use the proposed action itself as evidence for selecting
the control state.

Then determine whether the proposed action already realizes the inferred
control objective.

- If it does, preserve the proposed action.
```

```
- Otherwise, replace it with an executable action that better realizes the
  inferred mode and control target.

Mode-specific regulation:

EXPLORE:
Resolve the identified information gap before premature commitment.

EXPLOIT:
Make direct progress along the evidence-supported continuation.

RECOVER:
Repair, bypass, or leave the identified failed continuation and avoid simply
repeating the ineffective course.

Return JSON only:

{
  "mode": "EXPLORE|EXPLOIT|RECOVER",
  "evidence": ["concrete trajectory evidence"],
  "control_target": "one concise trajectory-specific objective",
  "action": "one executable environment action"
}
```

B.3 Fused Controller Prompt

The fused realization serializes proposal formation, control-state inference, and action regulation within a single structured completion. We reuse the task instructions, demonstrations, and interaction history of the base ReAct agent and append the following instruction:

```
[EER decision]

First form the next candidate action from the task and interaction history.

Then infer the active control state from the marked current task trajectory.

Control modes:

RECOVER:
Repair, bypass, or leave a continuation contradicted or rendered ineffective
by observed feedback.

EXPLORE:
Acquire a specific missing task-relevant fact when the current continuation
has not been invalidated.

EXPLOIT:
Advance a continuation already supported by trajectory evidence.

RECOVER takes precedence while an observed failure remains unresolved.

The evidence field must be grounded only in task, action, or observation
content contained in the marked current task trajectory.
Do not use demonstrations or unsupported assumptions as evidence.

Finally, determine whether the candidate action already realizes the inferred
control objective.

- If it does, preserve the candidate action.
```

```
- Otherwise, replace it with an executable action that better realizes the
  inferred mode and control target.

Return JSON only, with fields in the following order:

{
  "base_reasoning": "concise action reasoning",
  "proposal": "one candidate executable environment action",
  "mode": "EXPLORE|EXPLOIT|RECOVER",
  "evidence": ["concrete trajectory evidence"],
  "control_target": "one trajectory-specific control objective",
  "action": "one final executable environment action"
}
```

Only `action` is sent to the environment. The `proposal` and control-state fields are retained for analysis. The resulting observation, together with retained interaction records, is serialized into the trajectory available at the next decision step.

B.4 Controlled-Study and Ablation Prompts

We consider four controlled variants to isolate whether improvements arise from additional reasoning, access to the inferred control state, the mode-specific regulation operator, or the correctness of the inferred control structure.

Matched Reasoning. This condition matches E^2R with two additional language-model calls per actionable turn but removes the explicit control-state representation. The first call performs a generic trajectory assessment:

```
Historical trajectory:
{HISTORICAL TRAJECTORY}

Proposed action:
{PROPOSED ACTION}

Assess whether the proposed action remains appropriate given the trajectory
and identify any relevant issue. Do not propose the final action yet.
```

A second call then reconsiders the proposal using the assessment:

```
{BASE AGENT CONTEXT}

[Trajectory assessment]
{ASSESSMENT}

Proposed action:
{PROPOSED ACTION}

Based on the assessment, either preserve or revise the proposed action.
Output exactly one executable environment action. Do not output reasoning.
>
```

This condition controls for the possibility that performance gains arise simply from additional trajectory-level reasoning and model calls.

Mode Feedback. This condition uses the same trajectory-conditioned control-state inference as E^2R but removes explicit mode-conditioned regulation. Given τ_t, it first obtains

$$(m_t, e_t, \ell_t) = C(\tau_t). \tag{22}$$

The inferred state is then provided to the ordinary action generator as textual feedback rather than enacted through the E^2R regulator.

```
{BASE AGENT CONTEXT}

[Feedback]
Inferred control mode: {MODE}
Trajectory evidence:
{EVIDENCE}
Control target: {CONTROL TARGET}

The base agent proposed:
{PROPOSED ACTION}

Reconsider the proposed action accordingly using ordinary action generation.
Output exactly one executable environment action. Do not output reasoning.
>
```

The comparison with E^2R therefore tests whether explicitly enacting the inferred control state through a mode-specific operator provides benefit beyond merely exposing the same information to the language model.

Perturbed Control. This condition uses the same control pipeline as E^2R but perturbs the control state before regulation. At intervention steps, the inferred mode is replaced by one of the other two modes. Trajectory-grounded evidence and a corresponding control target are then constructed under the imposed alternative mode, after which the resulting state is enacted using the same regulation mechanism as E^2R.

This preserves explicit control enactment while altering the content of the control state, allowing us to test whether performance depends on alignment between the inferred trajectory state and the state that is enacted.

```
Historical trajectory:
{HISTORICAL TRAJECTORY}

The control mode is intentionally fixed to {PERTURBED MODE}.
Using only evidence contained in the trajectory, construct:
1. trajectory evidence consistent with this imposed mode;
2. a trajectory-specific control target under this mode.

Mode meanings:
- EXPLORE: acquire specific missing task-relevant information.
- EXPLOIT: make direct progress along an established continuation.
- RECOVER: repair, bypass, or leave a failed continuation.

Choose concrete trajectory details that make the imposed mode internally
coherent, without inventing observations. Generate a concise target for what
the next action should accomplish under that mode.

Return JSON only:
{
  "evidence": ["concrete trajectory evidence"],
  "control_target": "one concise sentence"
}
```

The resulting perturbed state is then enacted using the same mode-specific regulation operator as E^2R. This intervention preserves the regulation mechanism while corrupting the control variable itself, providing a direct test of whether the identity of the inferred control mode matters.

C Experimental Setup

C.1 Environment Details

Table 7 summarizes the evaluation splits and interaction budgets used in our experiments. Unless otherwise specified, all methods are evaluated on the same set of episodes within each environment. An episode terminates when the task is successfully completed, the environment signals termination, or the corresponding interaction budget is exhausted.

Table 7: Evaluation environments and interaction budgets. The environment-step budget specifies the maximum number of environment interactions allowed per episode.

Environment	Split / Subset	Episodes	Max Env. Steps	Max LLM Calls	Primary Metric
ALFWorld	eval_out_of_distribution	134	50	50	SR
WebArena-Lite	full benchmark	165	50	50	SR
ToolSandbox	non-RapidAPI scenarios	728	30	60	Score

ALFWorld. We use the complete eval_out_of_distribution split of ALFWorld, consisting of 134 household-interaction tasks. Each task requires the agent to complete a multi-step instruction through textual observations and discrete environment actions. We allow at most 50 environment interactions per episode and report binary task success using the environment's native success signal. All methods are evaluated on the same 134 tasks.

WebArena-Lite. We evaluate on the full WebArena-Lite benchmark introduced in VisualAgentBench (11), which refines 165 tasks from WebArena (32). The benchmark evaluates browser-based navigation, information retrieval, and interaction with stateful websites through multi-step browser actions.

We execute all 165 tasks through BrowserGym (8) using the WebArena high-level action space with `multiaction=False` and `strict=True`. Each episode is limited to 50 environment steps. Task completion is determined by the native evaluator, and we report success rate as the fraction of successfully completed tasks.

ToolSandbox. ToolSandbox evaluates stateful tool use under structured function-calling interfaces. We remove scenarios requiring the RapidAPI backend and evaluate on the resulting 728 scenarios. Unlike ALFWorld and WebArena-Lite, ToolSandbox provides a graded native evaluator rather than only a binary success indicator. We therefore use its native episode-level evaluation score as the primary performance metric. The agent is allowed at most 60 LLM calls per episode.

C.2 Metrics Details

Success Rate. For environments with binary task completion, namely ALFWorld and WebArena-Lite, we report success rate (SR). Let $y_i \in \{0, 1\}$ denote the final success indicator for episode i, and let N be the number of evaluation episodes. Then

$$\mathrm{SR} = \frac{1}{N} \sum_{i=1}^{N} y_i. \tag{23}$$

We report SR as a percentage in the main tables.

ToolSandbox Score. For ToolSandbox, we use the benchmark's native graded evaluation score. Let s_i denote the native evaluation score of episode i. The reported score is the macro-average over episodes,

$$\mathrm{Score} = \frac{1}{N} \sum_{i=1}^{N} s_i. \tag{24}$$

This aggregation assigns equal weight to every task, irrespective of its trajectory length.

Performance–Budget AUC. Final task performance alone does not distinguish an agent that reaches the solution quickly from one that obtains the same outcome only after substantially more interaction. We therefore additionally report *Performance–Budget AUC* (PB-AUC), which summarizes performance over the entire available interaction budget.

We use environment interactions as the budget variable. Specifically, let c_i denote the budget consumed by episode i, and let B be the maximum budget used to construct the performance–budget curve. The budget unit is environment steps for ALFWorld and WebArena-Lite, and the benchmark's effective turn count for ToolSandbox. All methods within the same environment use the same B.

Let v_i denote the terminal performance of episode i:

$$v_i = \begin{cases} y_i, & \text{for ALFWorld and WebArena-Lite,} \\ s_i, & \text{for ToolSandbox.} \end{cases} \tag{25}$$

We define the budget-dependent contribution of episode i as

$$p_i(b) = v_i \mathbb{I}[c_i \le b], \qquad b \in \{0, \dots, B\}. \tag{26}$$

Thus, an episode contributes zero before its terminal outcome is obtained and contributes its terminal performance at all subsequent budget points. For binary-success environments, this means that a successful episode contributes one from its completion step onward, whereas an unsuccessful episode contributes zero throughout.

The aggregate performance–budget curve is

$$P(b) = \frac{1}{N} \sum_{i=1}^{N} p_i(b). \tag{27}$$

We compute PB-AUC by trapezoidal integration over the integer-valued budgets:

$$\text{PB-AUC} = \frac{1}{B} \sum_{b=1}^{B} \frac{P(b-1) + P(b)}{2}. \tag{28}$$

Equivalently, this is the normalized area

$$\text{PB-AUC} = \frac{1}{B} \int_0^B P(b)\, \mathrm{d}b. \tag{29}$$

with linear interpolation between adjacent integer budget points. Because the underlying episode-level performance lies in $[0, 1]$, PB-AUC also lies in $[0, 1]$. A larger PB-AUC therefore indicates that an agent achieves useful task performance earlier and sustains it over a larger fraction of the available interaction budget.

C.3 Control-Behavior Metrics

For RQ2, we characterize how E^2R regulates the underlying base policy. Let T_i be the number of executed control steps in episode i, let $m_{i,t} \in \{\textsc{Explore}, \textsc{Exploit}, \textsc{Recover}\}$ be the inferred mode, and let

$$r_{i,t} = \mathbb{I}\left[\text{norm}(a_{i,t}) \neq \text{norm}(\hat{a}_{i,t})\right]. \tag{30}$$

indicate that the compatibility gate rejects the base proposal and the regulator redirects it before execution. Thus, $r_{i,t} = 1$ indicates that the action ultimately executed in the environment differs from the base proposal after environment-specific normalization.

Mode Usage. For each mode m, we first compute its fraction of control steps within each episode and then macro-average across episodes:

$$\text{Usage}(m) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{T_i} \sum_{t=1}^{T_i} \mathbb{I}[m_{i,t} = m]. \tag{31}$$

This episode-level averaging prevents long trajectories from dominating the reported mode distribution.

Redirect Rate. The overall Redirect Rate measures the mean within-episode fraction of control steps on which the executed action differs from the base proposal:

$$\text{RedirectRate} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T_i}\sum_{t=1}^{T_i} r_{i,t}. \tag{32}$$

Like Mode Usage, this statistic is macro-averaged across evaluated episodes, so each episode contributes equally regardless of its trajectory length. A low value indicates that the controller preserves most proposals and intervenes only selectively.

Intervention Coverage. Redirect Rate measures intervention frequency across steps, whereas Intervention Coverage measures how broadly those interventions are distributed across tasks. We define it as the fraction of eligible episodes containing at least one redirect:

$$\text{InterventionCoverage} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\left[\sum_{t=1}^{T_i} r_{i,t} > 0\right]. \tag{33}$$

Redirects per Episode. We additionally report the mean number of redirects per evaluation episode,

$$\text{Redirects/Episode} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i} r_{i,t}. \tag{34}$$

Together with Redirect Rate and Intervention Coverage, this distinguishes a controller that modifies many actions throughout every trajectory from one that makes sparse corrections concentrated in a subset of episodes.

Mode-Conditioned Redirect Rate. To examine whether intervention is associated with the inferred control state, we report the redirect frequency conditioned on each mode:

$$P(\text{Redirect} \mid m) = \frac{\sum_{i=1}^{N}\sum_{t=1}^{T_i}\mathbb{I}[m_{i,t}=m]\, r_{i,t}}{\sum_{i=1}^{N}\sum_{t=1}^{T_i}\mathbb{I}[m_{i,t}=m]}. \tag{35}$$

Unlike Mode Usage and the overall Redirect Rate, this conditional statistic uses a micro-averaged, step-pooled estimator: all eligible control steps assigned to mode m are pooled across episodes before the rate is computed.

D Control Behavior Analysis

E^2R organizes long-horizon interaction into temporally coherent control phases. As shown in Fig. 4, consecutive decisions remain in the same inferred mode 75.8–86.9% of the time, substantially above the 55.0–57.7% persistence obtained under episode-wise randomization that preserves mode counts. The structure of actual mode switches is environment dependent: ALFWorld predominantly transitions from Explore to Exploit, ToolSandbox exhibits a pronounced Exploit–Recover–Exploit repair pattern, whereas WebArena-Lite shows more heterogeneous switching. Together, these results indicate that E^2R organizes interaction into persistent control phases whose transition structure varies across environments, rather than following a fixed mode cycle.

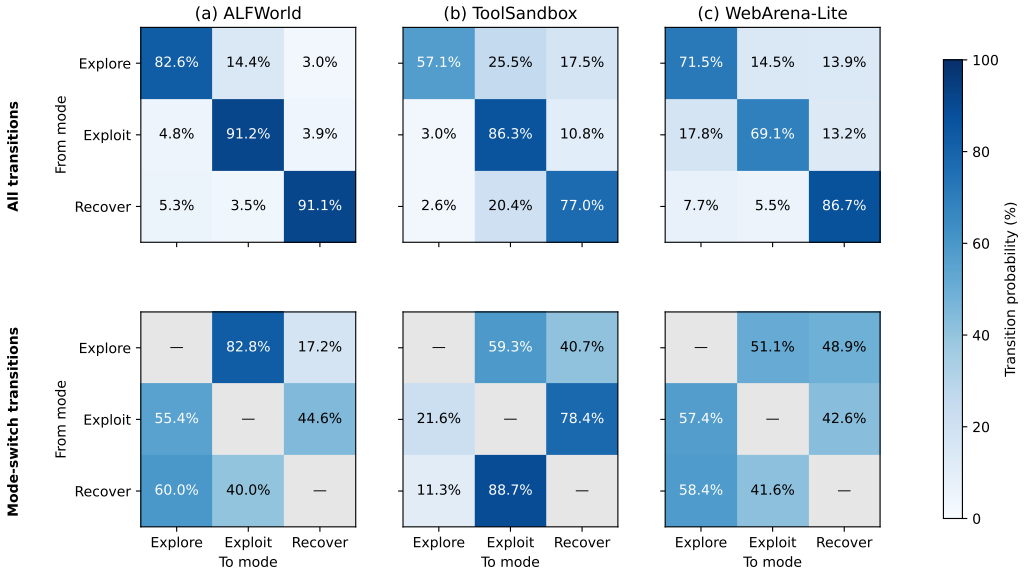


Figure 4: Temporal dynamics of E^2R's inferred control states. **Top:** Row-normalized transition probabilities $P(m_{t+1} \mid m_t)$, computed from consecutive controlled decisions pooled across episodes. The strong diagonal structure indicates that inferred modes tend to persist over contiguous interaction phases. Observed persistence is 86.9%, 75.8%, and 77.0% for ALFWorld, ToolSandbox, and WebArena-Lite, respectively, compared with 55.2%, 55.0%, and 57.7% under an episode-wise random-order null that preserves episode length and mode counts. **Bottom:** Switch-conditioned transition probabilities $P(m_{t+1} \mid m_t, m_{t+1} \neq m_t)$, with self-transitions removed and each row renormalized. Transition structure differs across environments: ALFWorld exhibits a pronounced Explore-to-Exploit transition, ToolSandbox shows an Exploit–Recover–Exploit repair pattern, while WebArena-Lite exhibits more heterogeneous switching among the three modes. The numbers of outgoing switches from (Explore, Exploit, Recover) are (186, 56, 40), (403, 213, 230), and (237, 176, 125) for ALFWorld, ToolSandbox, and WebArena-Lite, respectively.