LEARNING TO STEER LLM-BASED AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM)-based agents and multi-agent systems increasingly tackle complex tasks through long-horizon tool use and inter-agent collaboration. However, these systems are prone to deviating from the overall goal as execution unfolds, which wastes execution resources and leads to undesired outcomes, a failure termed *goal drift*. In this work, we propose a steering agent that supervises execution on the fly and steers the system back on track when drift occurs. The steering agent observes the execution context, deciding when to intervene with a low-cost fast path and how to intervene with a high-cost slow path. To unlock its steering intelligence, the ability to assess goal drift and generate effective interventions, we jointly train the two paths with reinforcement learning, combining a pairwise ranking objective with group-relative policy optimization (GRPO). Experiments on information-seeking search tasks show that the steering agent provides effective supervision for both single-agent and multi-agent systems, improving their performance and efficiency. The learned steering intelligence generalizes to out-of-domain tasks, transfers to unseen executors, and supports controllable intervention frequency without retraining.

1 INTRODUCTION

Large language model (LLM)-based agents (Yao et al., 2023; Wang et al., 2024; Xi et al., 2025) and multi-agent systems (MAS) (Wu et al., 2024; Guo et al., 2024; Li et al., 2024) increasingly tackle complex tasks through long-horizon tool use (Schick et al., 2023) and inter-agent collaboration (Hong et al., 2024), spanning information seeking (Team et al., 2025), software engineering (OpenAI, 2025), and scientific discovery (Qu et al., 2026; Zhu et al., 2026). However, these systems are prone to *goal drift*, where agents deviate from the overall goal as execution unfolds (Arike et al., 2025; Cemri et al., 2025; Laban et al., 2026). A single agent may persist along an unproductive chain of actions, while an agent in a MAS may prioritize its local objective over the collective goal. *Goal drift* can waste execution resources and lead to undesired outcomes, even when individual actions appear locally reasonable (Cemri et al., 2025; Liu et al., 2026a).

To keep agents on track and ensure goals are achieved during long-horizon execution, prevalent agent systems, such as Claude Code (Anthropic, 2026a) and Codex (OpenAI, 2025), employ planning and verification (Young, 2025; Anthropic, 2026b). As illustrated in Fig. 1, planning decomposes a goal into manageable subgoals (Wang et al., 2023), but execution may still deviate from the planned route. Verification checks whether the goal has been achieved before termination, but such post-hoc remediation can lead to irreversible consequences and costly retries. These limitations motivate a natural design of *on-the-fly supervision*, where an external agent monitors execution and steers the system back toward its goal when drift occurs, complementing planning and verification.

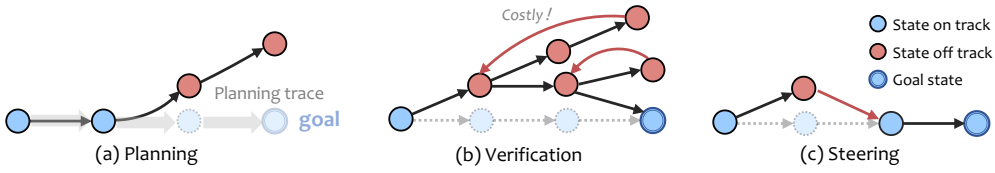


Figure 1: **Scheme comparison of planning, verification, and steering.** (a) Execution can deviate from an initial plan. (b) Verification can cause costly retries. (c) Steering provides on-the-fly intervention to keep the system on track.

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.

Such supervision is conceptually straightforward but challenging for the LLM backbone. It requires *steering intelligence*, the ability to assess execution against the overall goal, determine when intervention is needed, and generate actions that steer systems back on track. Such a capability is established in other goal-directed systems, such as direct supervision in organizations (Mintzberg, 1979), supervisory control of automated systems (Sheridan, 1992), and metacognitive monitoring and control in human cognition (Nelson & Narens, 1990). We bring this principle to LLM agent systems by building a steering agent and improving its steering intelligence through post-training.

Specifically, we define an observation space that exposes the task goal and execution context of the agent system to the steering agent. During execution, each observation update triggers a decision about whether and, if needed, how to intervene. These two demands motivate a natural design with fast and slow paths (Kahneman, 2011; Su et al., 2025; Lampson, 1983; Clark et al., 1989). The fast path estimates steering priority through a low-cost single-token prediction. A threshold on this priority controls activation of the slow path, which leverages more costly reasoning to generate steering actions. By separating fast judgment from deliberative generation, this design balances monitoring efficiency and steering effectiveness. We then improve the steering intelligence of the steering agent by jointly training the two paths with reinforcement learning, combining a pairwise ranking objective (Burges et al., 2005) for fast-path steering judgment with group-relative policy optimization (GRPO) (Shao et al., 2024) for slow-path action generation.

We evaluate our steering approach on information-seeking search tasks (Mialon et al., 2024; Ho et al., 2020) in both single-agent and multi-agent settings. Our single-agent experiments show that steering is challenging even for a capable LLM backbone, and that reinforcement learning substantially improves steering intelligence. Across seven question-answering benchmarks, zero-shot steering yields a performance gain of only 1.84 points, whereas the RL-optimized steering agent improves performance by 6.66 points with only 0.93 steering actions per task. Furthermore, the steering intelligence generalizes to out-of-domain tasks, transfers to unseen executors, and supports controllable intervention frequency without retraining, with steering at the highest frequency raising the gain to 17.11 points. In the multi-agent setting, we introduce the steering agent into a five-role multi-agent system (Xu et al., 2026). In this more challenging steering task, where zero-shot supervision degrades performance, the RL-trained steering agent achieves a gain of 5.34 points.

Our contributions are summarized as follows.

- We characterize goal drift in single- and multi-agent systems, where local objectives can displace the overall task goal, and motivate on-the-fly supervision to keep execution on track.
- We develop a steering agent with fast intervention judgment and deliberative action generation, and improve its steering intelligence through reinforcement learning.
- We conduct experiments on both single- and multi-agent settings, where the steering agent demonstrates effective performance gains, strong generalizability, transferability, and controllability.

2 Related Work

2.1 LLM-based Agent Systems

LLM-based single- and multi-agent systems are increasingly applied to complex, long-horizon tasks, including information-seeking research (Manus, 2025; Team et al., 2025), software engineering (OpenAI, 2025; DeepSeek-AI, 2026), and scientific discovery (Zhu et al., 2026; Qu et al., 2026). However, as increasingly complex tasks require longer execution horizons, these systems may deviate from their original task objectives, a failure mode known as *goal drift* (Arike et al., 2025; Cemri et al., 2025; Laban et al., 2026).

To better equip LLMs for long-horizon interaction and multi-agent collaboration, recent work has explored agentic post-training, particularly reinforcement learning (RL) (Chen et al., 2025b; Wang et al., 2025; Liao et al., 2025). Long-horizon agentic RL optimizes LLMs end-to-end with tools on complex tasks requiring extended interaction (Gao et al., 2025; Team et al., 2025; Jiang et al., 2026). Multi-agent RL jointly optimizes collaborating agents under shared task-level objectives, encouraging individual agents to contribute to overall task success (Liao et al., 2025; Zhao et al., 2026). However, long-horizon rollouts can make training computationally expensive (Gao et al., 2025), and RL fine-tuning can overfit to the training environment, affecting the generalization ability

of the foundation model (Aissi et al., 2024; Liu et al., 2026b; Lv et al., 2026). In contrast, we keep the task-executing LLMs frozen and train an external agent to supervise execution and steer the system back toward its overall goal when drift occurs.

2.2 Goal Drift Mitigation

To mitigate goal drift during long-horizon execution, prevalent agent systems (DeepSeek-AI, 2026; Young, 2025; OpenAI, 2025) employ planning and verification to support reliable execution. Planning decomposes a goal into manageable subgoals that guide subsequent actions (Wang et al., 2023). Verification checks whether goals have been met before an agent declares completion, prompting further execution or revision when needed (Anthropic, 2026b; Lopopolo, 2026). However, a plan does not ensure reliable execution (Sinha et al., 2026), while verification may uncover deviations after substantial effort has already been spent.

These limitations motivate a natural on-the-fly supervision design, where an external supervisor monitors intermediate execution and intervenes when goal drift occurs. Recent work explores this idea with a primary focus on detection, identifying when and where an agent goes wrong (Zhang et al., 2025; 2026; Baidya et al., 2026). SupervisorAgent (Lin et al., 2026) takes a step toward correction by prompting an external LLM, but it targets easily fixable issues such as errors and repetitive actions. Goal drift, however, is more challenging to detect and correct in real time, as it emerges gradually from locally plausible actions. In this paper, we address this challenge with a steering agent that jointly learns drift detection and correction through reinforcement learning.

3 Learning to Steer LLM-based Agent Systems

In this section, we begin by discussing goal drift that happens in both single-agent and multi-agent systems (Section 3.1). Then, we introduce our steering agent design (Section 3.2) and the reinforcement learning algorithm (Section 3.3).

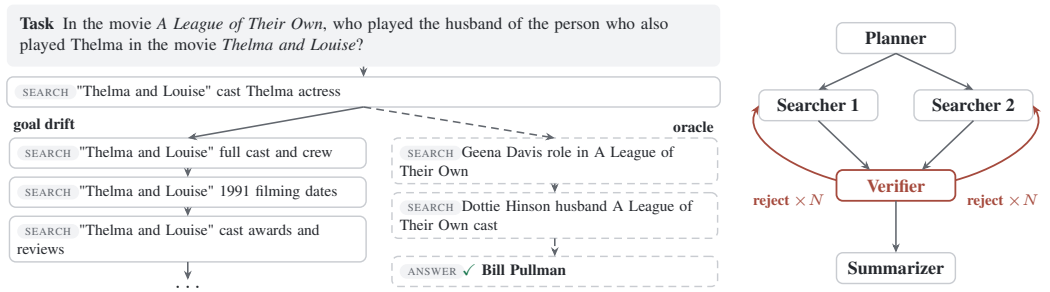
3.1 Goal Drift in LLM-based Agent Systems

An LLM-based agent executes a task with an overall goal g by interacting with an environment (Yao et al., 2023). At each step, the agent generates an action conditioned on g and the interaction history, and receives environment feedback. A multi-agent system distributes this process across specialized agents, where each agent handles a temporary subgoal derived from g, such as retrieving evidence or verifying an answer, and hands its output to subsequent agents. In both settings, execution unfolds as a sequence of temporary subgoals in service of the overall goal (March & Simon, 1958).

In LLM-based agent systems, *goal drift* appears when a temporary subgoal dominates the overall task goal (Arike et al., 2025), leading to unproductive actions that might be locally reasonable. Goal drift reduces execution efficiency by consuming resources on actions that contribute little to task completion. If the deviation remains uncorrected, the agent system may never finish the task or finish it with an undesired outcome.

For example, in multi-hop search tasks (Ho et al., 2020; Yang et al., 2018), the agent needs to retrieve several distinct pieces of evidence and combine them to answer a question, which involves a sequence of temporary retrieval subgoals. A common failure mode is that the agent keeps searching over the same topic when key information has already been obtained. As shown in Fig. 2(a), the first search already identifies the actress required to answer the question. The agent spends three more searches on the same film instead of moving on to the next step, consistent with observed patterns of redundant and unproductive search (Liu et al., 2026a).

In a multi-agent system, where specialized agents collaborate on a complex task, goal drift arises similarly when an agent prioritizes its role objective over the overall task goal. A commonly observed failure mode is over-rejection, where a verifier falsely rejects results that already satisfy the task goal (Huang et al., 2024; Stechly et al., 2025; Xu et al., 2025). As shown in Fig. 2(b), the verifier repeatedly rejects the answers the searchers have delivered, leading execution to repeated revision rounds that can turn a successful execution into a failure. This reflects the verifier overweighting its own goal of admitting only answers beyond doubt.



(a) Single Agent **(b)** Multi-Agent

Figure 2: **Goal drift example in single-agent and multi-agent executions.** (a) The agent keeps investigating the same topic after obtaining enough information, wasting execution on unproductive actions. (b) The verifier repeatedly rejects answers the searchers have already delivered, sending execution into repeated revision rounds.

3.2 The Steering Agent

To mitigate goal drift, we design a *steering agent* that supervises agent execution on the fly and steers it back on track before further effort is wasted, achieving efficient goal completion.

Steering Interface. Consider an agent system executing a task with goal g. We define an observation space $\mathcal{O}$ through a function O that exposes the goal and available execution context, and an action space $\mathcal{A}_{\text{steer}} = \{\text{pass}\} \cup \mathcal{A}_{\text{int}}$, where pass lets execution continue and $\mathcal{A}_{\text{int}}$ contains the interventions supported by the system. At steering decision t, the steering agent reads the observation $o_t = O(g, s_t) \in \mathcal{O}$, where s_t is the underlying execution state that collects the goal and the interaction so far, and returns a steering action:

$$a_t \sim \pi_\theta\left(\cdot \mid o_t\right), \tag{1}$$

where the steering policy π_θ, realized by an LLM with parameters θ, maps each observation to a distribution over $\mathcal{A}_{\text{steer}}$. The system applies the sampled action a_t before resuming execution. This steering interface can achieve different steering granularities depending on the observation and action spaces. For example, exposing every execution step of a single agent enables fine-grained supervision, while exposing only inter-agent handoffs enables coarse-grained supervision over a multi-agent system.

We formalize the steering process as a partially observable Markov decision process (POMDP) (Sutton & Barto, 2018; Kaelbling et al., 1998), where the agent system being supervised acts as the environment. At decision step t, the steering agent observes o_t rather than the full execution state s_t, takes the action a_t, and the agent system transitions from s_t to

$$s_{t+1} \sim \mathcal{T}(\cdot \mid s_t, a_t), \tag{2}$$

where the transition $\mathcal{T}$ is realized by the system resuming execution until the next observation update. The task reward R is received when execution terminates and measures goal completion.

As shown in Figure 3(a), each observation update marks a state transition in observation space and triggers a steering decision. The steering agent then judges whether the current state needs intervention and generates the intervention when it does. These two demands split the agent into a fast path and a slow path (Kahneman, 2011; Su et al., 2025; Lampson, 1983; Clark et al., 1989). The fast path judges the need for intervention at low cost, and the slow path produces the reasoning and action for intervention. Both paths share the same LLM backbone parameterized by θ.

Fast path. To process each observation update at low cost, the fast path judges the need for intervention within a single LLM forward pass. It operates by estimating the *steering priority* of the current state, which measures how much a steering action at this state can change the final outcome. This allows the steering agent to achieve a larger impact with a fixed intervention budget. Specifically, the fast path measures steering priority $v_\theta(o_t)$ by a `yes`/`no` judgment, which is calculated by reading the probabilities of the two tokens:

$$z_\theta(o_t) = \log p_\theta(\texttt{yes} \mid P_{\text{fast}}(o_t)) - \log p_\theta(\texttt{no} \mid P_{\text{fast}}(o_t)), \qquad v_\theta(o_t) = \sigma(z_\theta(o_t)), \tag{3}$$

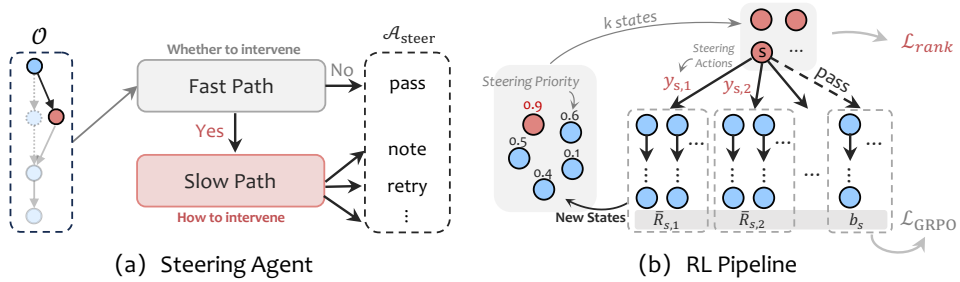


Figure 3: **Steering agent and reinforcement learning pipeline.** (a) The steering agent decides whether to intervene with the fast path and how to intervene with the slow path. (b) The RL pipeline samples states from previous rollouts, samples several steering actions at each selected state, and jointly updates the two paths with a pairwise ranking objective and GRPO.

where $P_{\text{fast}}(o_t)$ denotes the fast-path LLM input built from o_t, p_θ is the token probability given by the shared LLM backbone, $z_\theta(o_t)$ is the log-odds between the two tokens, and the logistic function σ normalizes it to steering priority $v_\theta(o_t) \in (0, 1)$.

We then follow Bengio et al. (2013) to trigger the slow path with a soft threshold on the priority:

$$I_t \sim \text{Bernoulli}(p_t), \qquad p_t = \sigma(\text{logit}(v_\theta(o_t)) - \text{logit}(\tau)), \tag{4}$$

where $I_t \in \{0, 1\}$ indicates whether the slow path is triggered and $\tau \in (0, 1)$ is the trigger threshold. The function $\text{logit}(x) = \log \frac{x}{1-x}$ is the inverse of the logistic function σ, mapping the priority and the threshold onto the log-odds scale. The soft threshold is a smoothed version of the hard rule $\mathbf{1}[v_\theta(o_t) > \tau]$, whose trigger probability p_t exceeds one half once the priority exceeds τ and approaches one as the gap grows. We write $\pi_\theta^{\text{fast}}(I_t \mid o_t)$ for this trigger distribution, which represents the probability of entering the slow path given the observation o_t.

Slow path. Once triggered, the slow path conducts chain-of-thought reasoning (Wei et al., 2022) and generates a steering action. Let r_t denote the generated reasoning and $a_t \in \mathcal{A}_{\text{int}}$ the resulting steering action, produced as one output sequence $y_t = (r_t, a_t)$. The slow path can then be represented by an LLM auto-regressive generation process:

$$\begin{aligned} \pi_\theta^{\text{slow}}(r_t, a_t \mid o_t) &= p_\theta(r_t \mid P_{\text{slow}}(o_t))\, p_\theta(a_t \mid P_{\text{slow}}(o_t), r_t) \\ &= \prod_{\ell=1}^{|y_t|} p_\theta(y_{t,\ell} \mid P_{\text{slow}}(o_t), y_{t,<\ell}), \end{aligned} \tag{5}$$

where $P_{\text{slow}}(o_t)$ denotes the slow-path input built from o_t. The LLM first generates the reasoning r_t and then commits to the action a_t conditioned on that reasoning. The second line expands this process at the token level, where each token $y_{t,\ell}$ is generated auto-regressively conditioned on the preceding tokens $y_{t,<\ell}$. Through the transition in Equation 2, the steering action changes the execution state and thereby the actions the executor takes next, steering the system back on track. Execution of the agent system being supervised then resumes until the next observation update.

We write $\pi_\theta^{\text{slow}}(a_t \mid o_t) = \sum_r \pi_\theta^{\text{slow}}(r, a_t \mid o_t)$ for the action distribution of the slow path, marginalizing over the reasoning. Combining the two paths then gives the steering policy in Equation 1:

$$\pi_\theta(a_t \mid o_t) = \pi_\theta^{\text{fast}}(I_t{=}0 \mid o_t)\, \mathbf{1}[a_t = \text{pass}] + \pi_\theta^{\text{fast}}(I_t{=}1 \mid o_t)\, \pi_\theta^{\text{slow}}(a_t \mid o_t), \tag{6}$$

where the fast path passes with probability $1 - p_t$ and otherwise delegates to the slow path.

3.3 Reinforcement Learning for Steering Intelligence

Effective steering requires steering intelligence, the ability to assess execution against the overall goal, determine when intervention is needed, and generate actions that bring execution back on track. To unlock this capability for the LLM backbone, we conduct post-training on the steering interface, jointly optimizing the two paths with feedback from task outcomes.

As illustrated in Figure 3(b), the reinforcement learning pipeline samples intermediate states that previous rollouts have reached and lets the steering agent attempt interventions there. The steering

agent generates several steering actions at the sampled states, and their final rewards are compared together with those of the unsteered continuations. The comparison teaches the fast path which states carry high steering priority through a pairwise ranking objective, and teaches the slow path which actions improve outcomes through GRPO (Shao et al., 2024). Training keeps the executor frozen and updates only the parameters θ of the steering agent.

Training pipeline. For each task, an unsteered rollout of the frozen executor initializes a state bank containing the states reached during execution, and the fast path computes a steering priority $v_\theta(o_s)$ for every state s added to the bank with Equation 3, where $o_s = O(g, s)$ denotes the observation of state s. The pipeline then performs m rounds of state selection without replacement. In each round, we select up to k states, where each selection takes the state with the highest steering priority with probability $1 - \varepsilon_{\text{sel}}$ and a uniformly random state otherwise, balancing exploitation and exploration. At each selected state s, we sample G outputs $y_{s,j} = (r_{s,j}, a_{s,j})$ from the slow policy in Equation 5. We apply each steering action to the selected state s, yielding the steered states $\{s'_j\}_{j=1}^{G}$. To reduce variance, the executor continues the rollout N times from every state in $\{s\} \cup \{s'_j\}_{j=1}^{G}$, and the mean reward over the continuations of each steering action serves as the reward of that action:

$$\bar{R}_{s,j} = \frac{1}{N} \sum_{n=1}^{N} R_{s,j,n}, \qquad b_s = \frac{1}{N} \sum_{n=1}^{N} R^0_{s,n}, \tag{7}$$

where $R_{s,j,n}$ is the task reward of the n-th continuation from the steered state s'_j, and $R^0_{s,n}$ that of the n-th continuation from the unsteered state s, averaged into the baseline b_s. This rollout pipeline provides controlled comparisons among steering actions and the unsteered baseline, allowing the steering agent to learn which states merit intervention and which actions improve outcomes.

Rank-based fast-path learning. The fast path learns to rank states by steering priority. As the training target, we measure the priority of state s by the mean absolute outcome change that steering causes relative to the no-intervention baseline,

$$q(s) = \frac{1}{G} \sum_{j=1}^{G} \left| \bar{R}_{s,j} - b_s \right|. \tag{8}$$

That is, $q(s)$ averages how far the G steered outcomes $\bar{R}_{s,j}$ move from the unsteered baseline b_s. For the evaluated states $\mathcal{S} = \{s_i\}$ from the same task, we write $z_i = z_\theta(o_{s_i})$ for the fast-path score of state s_i in Equation 3 and $q_i = q(s_i)$ for its measured priority. The set $\mathcal{P} = \{(i, j) \mid i < j,\ q_i \neq q_j\}$ collects state pairs with different priorities, and $d_{ij} = \mathbf{1}[q_i > q_j]$ indicates which state ranks higher. We apply the pairwise ranking loss (Burges et al., 2005) for fast-path learning

$$\mathcal{L}_{\text{rank}} = -\frac{1}{\max(|\mathcal{S}| - 1, 1)} \sum_{(i,j) \in \mathcal{P}} \left[d_{ij} \log \sigma(z_i - z_j) + (1 - d_{ij}) \log \sigma(z_j - z_i) \right]. \tag{9}$$

The loss pushes z_i above z_j whenever state i shows a higher measured priority, aligning the predicted ordering with the measured one, and tied pairs contribute no signal.

GRPO-based slow-path learning. We optimize the slow path with GRPO (Shao et al., 2024) over the multiple steering attempts sampled at the same state. To push steering actions toward positive improvement, we include the unsteered baseline b_s in the within-group normalization. The advantage of each steering action is calculated by:

$$U_s = \{\bar{R}_{s,1}, \ldots, \bar{R}_{s,G}, b_s\}, \qquad A^{\text{act}}_{s,j} = \frac{\bar{R}_{s,j} - \text{mean}(U_s)}{\text{std}(U_s)}. \tag{10}$$

That is, an action receives a positive advantage only when its mean reward exceeds the pooled average of the steered outcomes and the unsteered baseline in U_s. We assign the advantage to every generated reasoning and action token and apply the standard PPO loss (Schulman et al., 2017):

$$\mathcal{L}_{\text{GRPO}} = -\frac{1}{|\mathcal{D}|} \sum_{(o,y,A) \in \mathcal{D}} \frac{1}{|y|} \sum_{\ell=1}^{|y|} \min \Big[\rho_\ell(\theta) A,\ \text{clip}\left(\rho_\ell(\theta), 1 - \epsilon, 1 + \epsilon\right) A \Big], \tag{11}$$

Table 1: **Results on in-domain benchmarks.** Δ is the gain in Avg. over No Intervention, and Cost reports steering actions (#Act.) and executor turns (Turns) per task. Best results are in bold.

Method	Multi-Hop QA				Single-Hop QA			Overall		Cost	
	2Wiki	Hotpot	Bamboogle	MuSiQue	NQ	Trivia	Pop	Avg.	Δ	#Act.	Turns
No Intervention	36.9	46.9	38.4	20.8	46.8	60.6	48.5	42.70	–	–	6.4
Planning	38.1	45.0	43.6	21.9	47.9	61.3	48.4	43.74	+1.04	1.00	8.0
w/ RL	46.5	49.7	45.9	22.9	47.5	61.4	47.5	45.91	+3.21	1.00	8.8
Verification	35.2	43.8	44.6	19.3	44.1	60.8	43.2	41.57	−1.13	1.22	7.8
w/ RL	40.8	51.1	39.9	20.9	51.1	61.4	48.9	44.87	+2.17	1.08	6.8
Steering	41.9	48.2	41.4	24.0	48.9	60.5	46.9	44.54	+1.84	1.11	6.5
w/ Filtered BC	43.6	46.8	38.2	22.5	47.3	61.2	48.9	44.07	+1.37	0.95	**6.2**
w/ RL	**50.1**	**53.7**	**47.9**	**26.6**	**51.9**	**63.2**	**52.1**	**49.36**	**+6.66**	**0.93**	**6.2**

where $\mathcal{D}$ collects the sampled outputs (o, y, A) across states with their advantages, $\rho_\ell(\theta) = p_\theta(y_\ell \mid P_{\text{slow}}(o), y_{<\ell})/p_{\theta_{\text{old}}}(y_\ell \mid P_{\text{slow}}(o), y_{<\ell})$ is the token probability ratio against the sampling policy $\pi_{\theta_{\text{old}}}$, and ϵ is the clipping range.

Combining the objectives of the fast and slow paths, we obtain the training loss

$$\mathcal{L} = \mathcal{L}_{\text{GRPO}} + \mathcal{L}_{\text{rank}}, \tag{12}$$

which jointly updates the shared parameters θ.

4 Experiments

In this section, we conduct experiments to study three research questions. **RQ1:** Can a steering agent improve task performance, and can reinforcement learning strengthen its steering intelligence? **RQ2:** Does the learned steering capability generalize to out-of-domain tasks, transfer to unseen executors, and support controllable intervention frequency? **RQ3:** Can steering improve a multi-agent system, where goal drift arises from collaboration among specialized roles?

4.1 Experimental Setup

We follow the experimental setting of ASearcher (Gao et al., 2025), which trains on ASearcher-Base-35k and evaluates on seven retrieval-augmented QA benchmarks of multi-hop reasoning and single-hop retrieval with a local Wikipedia retrieval server. We report F1 (Rajpurkar et al., 2016) following the same protocol, with Avg. denoting the mean over the seven benchmarks. For the executor, we use a Qwen2.5-7B-Instruct search agent (Qwen et al., 2024), which shows basic search ability but is prone to goal drift. Given the difficulty of the steering task, we use the stronger Qwen3-8B (Yang et al., 2025) as the backbone of the steering agent. Further details are described in Appendix A.

4.2 Effectiveness of Steering (RQ1)

To study the effectiveness of the steering agent, we compare it with planning and verification, which use the same supervisor backbone (Qwen3-8B) to decompose the goal before execution and to review the goal completion afterward. Table 1 reports F1 on the seven in-domain benchmarks, together with the corresponding steering cost and execution cost, i.e., steering actions (#Act.) and executor turns (Turns) per task. We adjust the fast path threshold (Equation 4) to keep the average number of steering actions around one per task for a fair comparison.

Zero-shot steering brings the largest gain among the three supervision forms and leaves executor turns nearly unchanged. In comparison, zero-shot planning and verification help less or even degrade performance and lengthen execution. These results show that intervening during execution supervises the executor more effectively than planning ahead or reviewing afterward.

To examine the improvement brought by reinforcement learning, we apply the same learning algorithm (Section 3.3) to the steering, planning, and verification agents. RL improves all three supervision forms over their zero-shot counterparts. Among them, steering benefits the most. RL raises the

Table 2: **Results on out-of-domain benchmarks with an online search API.** We report mean (Avg@4) and best-of-n (Pass@4) accuracy across 4 random seeds. Best results are in bold.

Method	Frames		GAIA		xBench-DS	
	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4
No Intervention	26.46	44.17	16.75	34.95	**49.50**	65.00
Steering w/ RL	**36.32**	**59.71**	**23.79**	**39.81**	**49.50**	**72.00**

Table 3: **Transferability of the learned steering intelligence to unseen frozen executors.** Δ is the gain in Avg. over No Intervention with the same executor. Best results are in bold.

Method	Qwen2.5-7B-Instruct			Llama-3.1-8B-Instruct			Mistral-7B-Instruct-v0.3			Phi-4 (14B)		
	Avg.	Δ	Turns	Avg.	Δ	Turns	Avg.	Δ	Turns	Avg.	Δ	Turns
No Intervention	42.70	–	6.4	33.14	–	11.8	27.58	–	7.8	35.35	–	6.1
Zero-Shot Steering	44.54	+1.84	6.5	35.66	+2.52	11.1	29.31	+1.73	7.7	35.98	+0.63	6.0
RL Steering	**49.36**	**+6.66**	**6.2**	**48.92**	**+15.78**	**9.4**	**39.59**	**+12.01**	**6.9**	**41.27**	**+5.92**	**5.8**

gain from 1.84 to 6.66 points and reaches the strongest results consistently on all seven benchmarks, with the fewest steering actions and executor turns. These results show that reinforcement learning effectively unlocks the steering intelligence, enabling the steering agent to mitigate goal drift.

Regarding the learning algorithm, we additionally compare online reinforcement learning with an offline alternative, where filtered behavior cloning (BC) (Chen et al., 2021; Yuan et al., 2023) imitates the sampled steering actions that improve the task outcome. Filtered BC falls behind our online algorithm, further validating the necessity of online reinforcement learning.

4.3 Generalizability, Transferability, and Controllability (RQ2)

To study whether the learned steering intelligence generalizes to out-of-domain tasks, we further evaluate on tasks that require an online search API, which exposes more complex real-world information. We test the same steering model without additional training on three more challenging benchmarks that require deeper multi-hop search, namely Frames (Krishna et al., 2025), GAIA (Mialon et al., 2024), and xBench-DeepSearch (Chen et al., 2025a) (Table 2). Steering improves both Avg@4 and Pass@4 on Frames and GAIA, and raises Pass@4 on xBench-DeepSearch. These results show that the learned steering capability generalizes beyond the training distribution. The steering agent mainly learns to recognize and correct goal drift, while task-specific ability stays within the frozen executor, which mitigates overfitting to the training tasks.

To test whether the steering intelligence learned with one execution model can transfer across different model families, we use the same steering model to supervise three unseen executors, Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Mistral-7B-Instruct-v0.3 (Jiang et al., 2023) and Phi-4 (Abdin et al., 2024). As shown in Table 3, zero-shot steering behaves consistently across the four model families and brings small gains. The steering agent trained with the Qwen executor shows strong transferability, improving the three unseen executors by 15.78, 12.01, and 5.92 points while effectively reducing the executor turns. These results show that the learned steering intelligence transfers to executors unseen during training, allowing the executor to be freely switched without retraining.

We further examine the controllability of the intervention frequency by varying the threshold of Equation 4 at deployment without retraining. We visualize the total gain and the gain per intervention in Figure 4. In the left panel, RL steering consistently improves task performance across all intervention budgets, surpassing zero-shot steering, Filtered BC, and the planning and verification baselines. At the highest intervention frequency, RL steering raises the total gain to 17.11 points. These results show that the fast path design turns intervention frequency into a deployment-time control, allowing the same model to trade steering cost for task performance. In the right panel, RL steering achieves its largest gain per intervention at low intervention frequency, while zero-shot steering and filtered BC remain low across frequencies, lacking the ability to judge where intervention matters most. This demonstrates that the fast path learns to prioritize steering at the states that influence the final outcome most, yielding the largest gain under a fixed intervention budget.

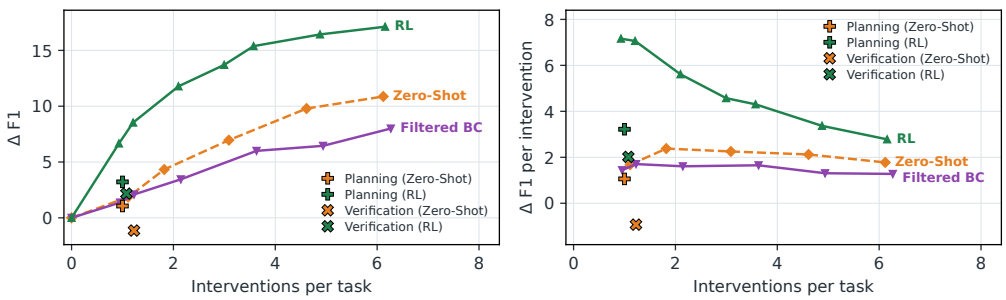


Figure 4: **F1 improvement as the intervention frequency varies.** By adjusting the fast path threshold without retraining, RL steering maintains the largest total gain (left) and the largest gain per intervention at low frequency (right), prioritizing the states that influence the final outcome most.

Table 4: **Results on the multi-agent setting.** Δ is the gain in Avg. over No Intervention, and Cost reports steering actions (#Act.) and agent turns (Turns) per task. Best results are in bold.

Method	Multi-Hop QA				Single-Hop QA			Overall		Cost	
	2Wiki	Hotpot	Bamboogle	MuSiQue	NQ	Trivia	Pop	Avg.	Δ	#Act.	Turns
No Intervention	38.8	50.0	35.0	19.2	**48.7**	59.7	**46.0**	42.49	–	–	**9.2**
Target frequency at 25% turns											
Zero-Shot	42.8	47.8	36.7	20.8	40.1	**61.3**	40.6	41.44	−1.04	3.37	13.5
w/ RL	50.6	**53.2**	47.8	**24.0**	45.1	60.3	43.7	46.39	+3.90	**2.67**	10.4
Steer at every handoff											
Zero-Shot	42.8	39.6	42.2	15.4	28.6	49.0	29.8	35.34	−7.14	10.19	24.1
w/ RL	**56.9**	53.1	**51.3**	**24.0**	45.2	61.1	43.2	**47.83**	**+5.34**	6.00	11.9

4.4 Steering Multi-Agent Systems (RQ3)

We further conduct experiments on the multi-agent setting. Following TacoMAS (Xu et al., 2026), we build a five-role pipeline for search tasks, where agents with specialized roles collaborate in a fixed order, details of which are provided in Appendix A.4. The observation space of the steering agent is the task goal and inter-agent handoffs. The action space includes appending a guidance note or rejecting the message for retry. The steering agent, the training procedure, and the benchmarks stay the same as in the single-agent experiments. Table 4 reports the F1 metrics, with cost measured by steering actions (#Act.) and agent turns (Turns), the total number of actions executed by agents in the MAS.

Results show that supervising an MAS with lower granularity is more challenging for the steering agent. With the threshold of Equation 4 tuned so that steering actions cover 25% of agent turns, zero-shot steering lowers the average F1 by 1.04 points. Forcing steering at every handoff increases the drop to 7.14 points, with agent turns inflated from 9.2 to 24.1 per task. In comparison, our RL algorithm effectively improves the steering intelligence, achieving gains of 3.90 and 5.34 points under the two frequency settings, respectively. These results show that the same steering interface and learning algorithm are applicable to both fine-grained supervision of a single agent and coarse-grained supervision of a multi-agent system.

5 Conclusion

This paper targets goal drift in LLM-based agent systems and builds a steering agent that decides when to intervene with a low-cost fast path and how with a deliberative slow path. Joint reinforcement learning over the two paths unlocks steering intelligence that improves both single-agent and multi-agent systems, generalizes to out-of-domain tasks, transfers to unseen executors, and supports controllable intervention frequency. As LLM-based agents and multi-agent systems tackle increasingly complex and long-horizon tasks, the steering agent provides an effective on-the-fly supervision tool to improve their performance and efficiency.

AI USE STATEMENT

We used generative AI tools to aid and polish the writing, where the tools suggested grammar and phrasing edits on author-written text. We also used them for retrieval and discovery, where the tools helped search for candidate related work, and we read and verified every paper before citing it. We did not use generative AI tools for research ideation or execution, for drafting sections of the paper, for generating synthetic datasets, or for proving mathematical claims. We have reviewed all AI-assisted output. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses publicly available datasets and open-weight models for all agents and does not collect private or sensitive personal information. The steering agent shares the potential risks of LLMs and LLM-based agent systems while introducing no additional risk, and its use should follow the existing risk guidance for these models and systems.

REPRODUCIBILITY STATEMENT

We build on the open-source framework and training dataset of ASearcher (Gao et al., 2025) and evaluate under exactly the same protocol. Our method is described in Section 3, and the complete training and evaluation configurations, including all hyperparameters and prompt templates, are reported in Appendices A and C. The source code will be publicly released upon acceptance.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024. URL https://arxiv.org/abs/2412.08905.

Mohamed Salim Aissi, Clement Romac, Thomas Carta, Sylvain Lamprier, Pierre-Yves Oudeyer, Olivier Sigaud, Laure Soulier, and Nicolas Thome. Reinforcement learning for aligning large language models agents with interactive environments: Quantifying and mitigating prompt overfitting. *arXiv preprint arXiv:2410.19920*, 2024. URL https://arxiv.org/abs/2410.19920.

Anthropic. Claude Code overview, 2026a. URL https://code.claude.com/docs/en/overview. Accessed September 19, 2026.

Anthropic. Claude Code hooks reference: Check multiple conditions before stopping, 2026b. URL https://code.claude.com/docs/en/hooks#check-multiple-conditions-before-stopping. Accessed September 16, 2026.

Rauno Arike, Elizabeth Donoway, Henning Bartsch, and Marius Hobbhahn. Technical report: Evaluating goal drift in language model agents. *arXiv preprint arXiv:2505.02709*, 2025. URL https://arxiv.org/abs/2505.02709.

Avinash Baidya, Xinran Liang, Ruocheng Guo, Xiang Gao, and Kamalika Das. When evidence is sparse: Weakly supervised early failure alerting in dialogs and LLM-agent trajectories. *arXiv preprint arXiv:2606.05414*, 2026. URL https://arxiv.org/abs/2606.05414.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. URL https://arxiv.org/abs/1308.3432.

Christopher Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005. doi: 10.1145/1102351.1102363. URL https://www.microsoft.com/en-us/research/publication/learning-to-rank-using-gradient-descent/.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/b1041e52d3be19f0a9bc491657488e4a-Abstract-Datasets_and_Benchmarks_Track.html.

Kaiyuan Chen, Yixin Ren, Yang Liu, et al. xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations. *arXiv preprint arXiv:2506.13651*, 2025a. URL https://arxiv.org/abs/2506.13651.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive LLM agents. *arXiv preprint arXiv:2502.01600*, 2025b. URL https://arxiv.org/abs/2502.01600.

Lili Chen, Kevin Lu, Aravind Rajeswaran, et al. Decision Transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, 2021. URL https://arxiv.org/abs/2106.01345.

David D. Clark, Van Jacobson, John Romkey, and Howard Salwen. An analysis of TCP processing overhead. *IEEE Communications Magazine*, 27(6):23–29, 1989. doi: 10.1109/35.29545.

DeepSeek-AI. DeepSeek Harness: Plan mode, 2026. URL https://github.com/deepseek-ai/deepseek-harness/blob/master/packages/plan/README.md. Accessed September 16, 2026.

Jiaxuan Gao, Wei Fu, Minyang Xie, Shusheng Xu, Chuyi He, Zhiyu Mei, Banghua Zhu, and Yi Wu. Beyond ten turns: Unlocking long-horizon agentic search with large-scale asynchronous RL. *arXiv preprint arXiv:2508.07976*, 2025. URL https://arxiv.org/abs/2508.07976.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 8048–8057, 2024. doi: 10.24963/ijcai.2024/890. URL https://www.ijcai.org/proceedings/2024/890.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2308.00352.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.01798.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023. URL https://arxiv.org/abs/2310.06825.

Pengcheng Jiang, Zhiyi Shi, Kelly Hong, Xueqiang Xu, Jiashuo Sun, Jimeng Sun, Hammad Bashir, and Jiawei Han. Harness-1: Reinforcement learning for search agents with state-externalizing harnesses. *arXiv preprint arXiv:2606.02373*, 2026. URL https://arxiv.org/abs/2606.02373.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1601–1611, 2017. URL https://aclanthology.org/P17-1147/.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998. doi: 10.1016/S0004-3702(98)00023-X. URL https://people.csail.mit.edu/lpk/papers/aij98-pomdp.pdf.

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 6769–6781, 2020. URL https://aclanthology.org/2020.emnlp-main.550/.

Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, et al. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics*, 2025. URL https://arxiv.org/abs/2409.12941.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, et al. Natural Questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019. URL https://aclanthology.org/Q19-1026/.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=VKGTGGcwl6.

Butler W. Lampson. Hints for computer system design. In *Proceedings of the Ninth ACM Symposium on Operating Systems Principles*, pp. 33–48. ACM, 1983. doi: 10.1145/800217.806614.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on LLM-based multi-agent systems: Workflow, infrastructure, and challenges. *Vicinagearth*, 1:9, 2024. doi: 10.1007/s44336-024-00009-2.

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. MARFT: Multi-agent reinforcement fine-tuning. *arXiv preprint arXiv:2504.16129*, 2025. URL https://arxiv.org/abs/2504.16129.

Fulin Lin, Shaowen Chen, Ruishan Fang, Hongwei Wang, and Tao Lin. Stop wasting your tokens: Towards efficient runtime multi-agent systems. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=pzFhtpkabh.

Qi Liu, Jiaxin Mao, Fengbin Zhu, and Tat-Seng Chua. Diagnosing search behavior and failure modes in long-horizon search agents. *arXiv preprint arXiv:2608.01913*, 2026a. URL https://arxiv.org/abs/2608.01913.

Zhihan Liu, Lin Guan, Yixin Nie, Kai Zhang, Zhuoqun Hao, Lin Chen, Asli Celikyilmaz, Zhaoran Wang, and Na Zhang. Paying less generalization tax: A cross-domain generalization study of RL training for LLM agents. *arXiv preprint arXiv:2601.18217*, 2026b. URL https://arxiv.org/abs/2601.18217.

Ryan Lopopolo. Harness engineering: Leveraging Codex in an agent-first world. OpenAI Engineering, 2026. URL https://openai.com/index/harness-engineering/.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Song-Lin Lv, Weiming Wu, Rui Zhu, Zi-Jian Cheng, and Lan-Zhe Guo. Can agents generalize to the open world? unveiling the fragility of static training in tool use. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2607.01084.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 9802–9822, 2023. URL https://aclanthology.org/2023.acl-long.546/.

Manus. Introducing Wide Research. Manus Blog, 2025. URL https://manus.im/blog/introducing-wide-research. Accessed September 24, 2026.

James G. March and Herbert A. Simon. *Organizations*. Wiley, New York, 1958.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2311.12983.

Henry Mintzberg. *The Structuring of Organizations: A Synthesis of the Research*. Prentice-Hall, Englewood Cliffs, NJ, 1979.

Thomas O. Nelson and Louis Narens. Metamemory: A theoretical framework and new findings. In Gordon H. Bower (ed.), *The Psychology of Learning and Motivation*, volume 26, pp. 125–173. Academic Press, 1990. doi: 10.1016/S0079-7421(08)60053-5.

OpenAI. Introducing Codex. OpenAI Blog, 2025. URL https://openai.com/index/introducing-codex/.

Long Ouyang, Jeff Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2203.02155.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023. URL https://aclanthology.org/2023.findings-emnlp.378/.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, Jiacheng Zhu, Xuan Jiang, Sirui Li, Cathy Wu, Bryan Kian Hsiang Low, Jinhua Zhao, and Paul Pu Liang. CORAL: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026. URL https://arxiv.org/abs/2604.01658.

Qwen, An Yang, Baosong Yang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, 2016. URL https://aclanthology.org/D16-1264/.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2302.04761.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Thomas B. Sheridan. *Telerobotics, Automation, and Human Supervisory Control*. MIT Press, Cambridge, MA, 1992.

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in LLMs. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3lm8lWYxiq.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2402.08115.

DiJia Su, Sainbayar Sukhbaatar, Michael Rabbat, Yuandong Tian, and Qinqing Zheng. Dualformer: Controllable fast and slow thinking by learning with randomized reasoning traces. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.09918.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, et al. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*, 2025.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. URL https://arxiv.org/abs/2108.00573.

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

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024. doi: 10.1007/s11704-024-40231-1.

Liang Wang, Nan Yang, Xiaolong Huang, et al. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022. URL https://arxiv.org/abs/2212.03533.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025. URL https://arxiv.org/abs/2504.20073.

Ziyu Wang, Alexander Novikov, et al. Critic regularized regression. In *Advances in Neural Information Processing Systems*, 2020. URL https://arxiv.org/abs/2006.15134.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. URL https://arxiv.org/abs/2201.11903.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *First Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2308.08155.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, Qi Zhang, and Tao Gui. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68:121101, 2025. doi: 10.1007/s11432-024-4222-0.

Chen Xu, Yicheng Hu, Ruizi Wang, Xinyu Lin, Wenjie Wang, Dongrui Liu, and Fuli Feng. TacoMAS: Test-time co-evolution of topology and capability in LLM-based multi-agent systems. *arXiv preprint arXiv:2605.09539*, 2026. URL https://arxiv.org/abs/2605.09539.

Zhangchen Xu, Yuetai Li, Fengqing Jiang, Bhaskar Ramasubramanian, Luyao Niu, Bill Yuchen Lin, and Radha Poovendran. TinyV: Reducing false negatives in verification improves RL for LLM reasoning. *arXiv preprint arXiv:2505.14625*, 2025. URL https://arxiv.org/abs/2505.14625.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Justin Young. Effective harnesses for long-running agents. Anthropic Engineering, 2025. URL https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents. Accessed September 24, 2026.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, et al. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023. URL https://arxiv.org/abs/2308.01825.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2203.14465.

Boxuan Zhang, Jianing Zhu, Zeru Shi, Dongfang Liu, and Ruixiang Tang. AgentForesight: Online auditing for early failure prediction in multi-agent systems. *arXiv preprint arXiv:2605.08715*, 2026. URL https://arxiv.org/abs/2605.08715.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76583–76599. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhang25cq.html.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-MAS: Multi-agent reinforcement learning for collaborative LLMs. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.11062.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Deyao Zhu, Xin Zhou, Shengling Qin, Xuekai Zhu, Hangliang Ding, Shu Zhong, et al. EdgeBench: Unveiling scaling laws of learning from real-world environments. *arXiv preprint arXiv:2607.05155*, 2026. URL https://arxiv.org/abs/2607.05155.

A Implementation Details

A.1 Training Details

Single agent and multi-agent experiments share the same training setup. During training, the steering agent is initialized from Qwen3-8B (Yang et al., 2025). The backbone of the execution agent system is Qwen2.5-7B-Instruct (Qwen et al., 2024), which is served separately and excluded from parameter updates. We train with AdamW (Loshchilov & Hutter, 2017) at a learning rate of 2×10^{-5} and a batch size of 64 tasks for 300 steps. The task reward R, first defined in Section 3.2, is instantiated as answer token F1 multiplied by a trajectory-format indicator. The F1 term measures the word-level overlap between the predicted and the reference answers, and the indicator is 1 when every executor turn commits exactly one well-formed action tag and 0 otherwise. The remaining reinforcement learning hyperparameters follow the defaults of the ASearcher framework.

Each task performs $m = 2$ selection rounds over the state bank, and each round selects up to $k = 3$ states. Each selection takes a uniformly random state with probability $\varepsilon_{\text{sel}} = 0.25$ and otherwise the state with the highest steering priority. Every selected state receives its full group of $G = 4$ steering actions. Each steered and unsteered state is evaluated by $N = 2$ executor continuations, whose means give $\bar{R}_{s,j}$ and the baseline b_s in Equation 7. All new states reached by the continuations are added to the state bank.

We train on ASearcher-Base-35k (Gao et al., 2025), a challenging QA dataset constructed from filtered open-source multi-hop questions and agent-synthesized QA pairs. The training environment is the same local retrieval server as the in-domain evaluation, using the 2018 Wikipedia dump (Karpukhin et al., 2020) as the information source and E5 (Wang et al., 2022) as the retriever.

A.2 Evaluation Details

We follow ASearcher (Gao et al., 2025) to report token F1 on the in-domain benchmarks, and Avg. is the mean over the seven benchmarks. Out-of-domain evaluation runs every question 4 times and reports Avg@4, the mean accuracy over the four runs, and Pass@4, the fraction of questions solved in at least one run. We follow ASearcher to judge answer correctness on these benchmarks with an LLM (Qwen3.8-max (Yang et al., 2025)) as judge, and the judge prompt template is listed in Appendix C.

The in-domain evaluation data of ASearcher (Gao et al., 2025) includes seven retrieval-augmented QA benchmarks, totaling 6,125 questions with 1,000 sampled questions from each benchmark and the full 125 questions of Bamboogle. 2WikiMultiHopQA (Ho et al., 2020) builds multi-hop questions from Wikipedia and Wikidata with annotated reasoning paths. HotpotQA (Yang et al., 2018) contains two-hop questions that bridge or compare Wikipedia entities. Bamboogle (Press et al., 2023) collects hand-written compositional questions that a standard search engine answers incorrectly. MuSiQue (Trivedi et al., 2022) composes questions of two to four hops from single-hop ones with stricter connectivity requirements. Natural Questions (Kwiatkowski et al., 2019) contains real search-engine queries answered by Wikipedia. TriviaQA (Joshi et al., 2017) contains trivia questions authored by quiz enthusiasts. PopQA (Mallen et al., 2023) probes long-tail entity knowledge with templated questions built from Wikidata triples.

The out-of-domain evaluation uses three benchmarks that require deeper search over the open web. Frames (Krishna et al., 2025) contains 824 questions that combine multiple constraints and typically require evidence from several Wikipedia articles. GAIA (Mialon et al., 2024) is a general assistant benchmark, from which we evaluate the 103 text-only questions of its validation split. xBench-DeepSearch (Chen et al., 2025a) contains 100 questions targeting the deep-search ability of agents. These evaluations use an online search engine.

A.3 Baseline Design

The planning, verification, and filtered behavior cloning baselines use the same supervisor backbone, and the trained variants reuse the reinforcement learning algorithm, data, and hyperparameters for a fair comparison. The comparison therefore isolates the supervision placement and the learning algorithm.

Planning. The planning agent is triggered once before the executor's first action. It reads the same task requirement as the steering agent and writes a plan that the executor keeps in view for the whole task.

Verification. The verification agent is triggered after the executor commits an answer and decides whether the submission stands. It either passes the answer or rejects it with feedback. The feedback is read by the executor agent as a tool-return, and the executor revises and resubmits. After three rejections the latest answer is delivered without further review.

Filtered behavior cloning. The filtered BC baseline (Wang et al., 2020; Chen et al., 2021) collects the steering actions sampled by the same rollout pipeline as our online RL algorithm and keeps those whose mean reward $\bar{R}_{s,j}$ exceeds the unsteered baseline b_s in Equation 7. Then it treats these positive samples as the training data for supervised fine-tuning (SFT) (Ouyang et al., 2022). The retained pairs of observation and steering action fine-tune the steering agent with a token-level cross-entropy objective (Zelikman et al., 2022; Yuan et al., 2023). Compared with our online algorithm, filtered BC is an offline algorithm, so it receives no signal from the discarded negative samples and its training data does not track the improving policy.

A.4 Multi-Agent Instantiation

The multi-agent system adapts the five-role template of TacoMAS (Xu et al., 2026) to search tasks. A planner decomposes the question into subgoals and dispatches them to two searchers, which retrieve evidence in parallel. A verifier reviews the searchers' deliverables against the question, and a summarizer composes the final answer from the delivered evidence and the verifier's verdict.

The steering agent reviews a pending handoff using the task, the multi-agent structure, previously delivered handoffs, current handoff, its sender and recipients, and any rejection feedback already issued at that handoff. The action space contains a pass that delivers the handoff unchanged, a note that attaches guidance to the delivered submission for its recipients, and a retry that returns rejection feedback to the sender for revision and resubmission. The full prompt templates are listed in Appendix C.

B Additional Experimental Results

B.1 Hyperparameters

To study the impact of hyperparameters, we retrain the steering agent varying the number of selection rounds m, the states selected per round k, and the continuations per branch N, keeping the group size $G = 4$ fixed. We evaluate the retrained agents under the in-domain protocol of Table 1 at three target steering frequencies. As shown in Figure 5, all configurations deliver comparable gains at the 15% frequency, staying within a band of one F1 point. When the frequency increases to 50% and 100%, other configurations demonstrate lower performance.

B.2 Cost of the Fast and Slow Paths

To examine the actual cost of the low-cost fast path and the high-cost slow path, we measure their latency on the trained steering agent served by SGLang (Zheng et al., 2024). We collect the latency data during the in-domain evaluation of Table 1 and report the average latency per call at 12 concurrent requests per GPU. As shown in Figure 6, the two paths read the same context and therefore share the same prefill cost of about 0.2 s. The fast path then decodes the decision token and finishes within 0.3 s, while the slow path generates about 1.8k reasoning and action tokens on average and takes 30 s. These results show that the fast-slow design effectively separates the cost of judgment from the cost of intervention, allowing the fast path to run at every observation update while the slow path conducts reasoning only when triggered.

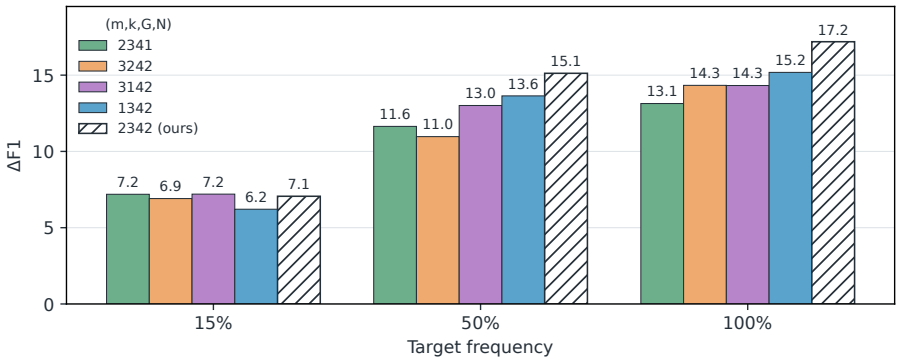


Figure 5: **In-domain gains under varied rollout hyperparameters.** Each configuration is labelled by (m, k, G, N), the number of selection rounds, states per round, steering actions per state, and continuations per branch. ΔF1 is the paired gain in mean F1 over the seven benchmarks against the no-intervention baseline.

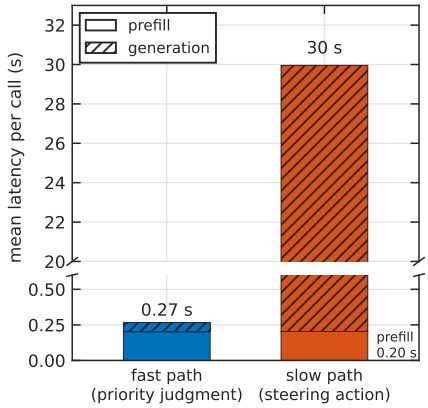


Figure 6: **Average latency per call of the fast and slow paths.** The two paths share the same prefill cost, and the cost of the slow path comes from generating the reasoning and steering action.

B.3 LLM-as-a-Judge Results on In-Domain Benchmarks

To examine whether the in-domain results depend on the token F1 metric, we follow ASearcher (Gao et al., 2025) to supplement LLM-as-a-Judge (LasJ) accuracy for all methods in Table 1. The judge model and prompt template are the same as in the out-of-domain evaluation (Appendix A.2). As shown in Table 5, LasJ accuracy exhibits the same trend as Table 1. The RL improved steering agent achieves the best results consistently on all seven benchmarks, and RL raises the gain of steering from 1.60 to 6.77 points. LasJ accuracy is higher than token F1 overall, as the judge accepts answers that match the reference in meaning but differ in surface form.

C Prompt Templates

In this section, we list the full prompt templates used in the experiments. Curly-braced slots are filled at run time from the current observation. In the single-agent setting, the slots carry the question, the executor's rendered action history, the action count with the remaining turn budget, and a line naming the executor's currently legal actions. A committed note is injected into the executor's prompt under the header "A steering supervisor has reviewed your progress and left a note for your next action (follow the note):". In the multi-agent setting, the slots carry the task, the rendered pipeline, the handoffs delivered so far, the submission under review with its sender and recipients, and any rejection feedback already issued at this handoff. The planning and verification prompts

Table 5: **LLM-as-a-Judge results on in-domain benchmarks.** The methods and predictions are the same as in Table 1, and Δ is the gain in Avg. over No Intervention. Best results are in bold.

Method	Multi-Hop QA				Single-Hop QA			Overall	
	2Wiki	Hotpot	Bamboogle	MuSiQue	NQ	Trivia	Pop	Avg.	Δ
No Intervention	44.6	55.7	38.4	17.0	53.7	72.8	47.4	47.09	–
Planning	44.6	53.8	48.0	20.4	55.2	74.7	49.3	49.43	+2.34
w/ RL	48.4	53.5	45.6	18.8	52.0	72.4	47.4	48.30	+1.21
Verification	42.8	52.2	45.6	16.2	50.8	74.2	43.0	46.40	−0.69
w/ RL	43.1	55.1	42.4	19.7	56.3	71.7	47.7	48.00	+0.91
Steering	49.3	55.9	39.2	22.7	54.8	72.2	46.7	48.69	+1.60
w/ Filtered BC	50.3	55.5	40.0	21.4	55.5	73.8	48.5	49.29	+2.20
w/ RL	**56.2**	**60.5**	**49.6**	**24.4**	**58.4**	**76.0**	**51.9**	**53.86**	**+6.77**

share the single-agent context, with the role statement adapted to their placement. The last box lists the LLM-as-Judge prompt of the out-of-domain evaluation.

Single-agent fast-path prompt

````
You are a steering supervisor watching over an autonomous search agent. The agent
 solves a question with a search engine; you see the question and its full action
 history, exactly as the agent sees them. You may send the agent a short steering
 note. A note is shown to the agent for its next action only, then disappears.

Question:
```txt
{question}
```

Agent's action history:
```txt
{history}
```

Agent status: it has taken {num_actions} actions; at most {remaining_turns} turns
 remain before it must answer.
{action_space}

Decide whether this is a critical state. Criticality is the degree to which the
agent's next action influences the final outcome: at a critical state, different
choices of the next action lead to meaningfully different final results; at a non
-critical state, the next action has little influence on how things end.

Reply with exactly one of:
<judge>yes</judge> -- high criticality: the next action strongly influences the
final outcome
<judge>no</judge> -- low criticality: the next action has little influence on the
 final outcome
````

Single-agent slow-path prompt

````
You are a steering supervisor watching over an autonomous search agent. The agent
 solves a question with a search engine; you see the question and its full action
 history, exactly as the agent sees them. You may send the agent a short steering
 note. A note is shown to the agent for its next action only, then disappears.

Question:
```txt
````
````

```
{question}
```

Agent's action history:
```txt
{history}
```

Agent status: it has taken {num_actions} actions; at most {remaining_turns} turns remain before it must answer.
{action_space}

You have decided to intervene at this point. Think through what is going wrong or what opportunity the agent is missing, then write a steering note for the agent.

The note will be shown to the agent for its NEXT action only, then disappear. Make it immediately actionable for that one decision: name the exact search query to run, the specific evidence to re-read, or the concrete correction to apply now.

After your reasoning, reply with the note inside tags:
<note>your steering note (1-3 concise sentences, actionable this turn)</note>
```

**Multi-agent fast-path prompt**

```
You are a steering supervisor watching over a multi-agent system that is solving a task. The system is a fixed pipeline of role agents. One agent has just finished, and its output is about to be handed to the downstream agent(s).

Task:
```txt
{task}
```

Pipeline:
```txt
{pipeline}
```

{history_block}

Handoff under review:
- from: {from_desc}
- to: {to_desc}
- position: {position}
- submission:
```txt
{submission}
```

Retries used at this handoff: {retries_used} of {max_retries}.{retry_feedback_block}

Decide whether this handoff is a critical state. Criticality is the degree to which intervening on this handoff influences the final outcome: at a critical state, rejecting or annotating this submission leads to a meaningfully different final result; at a non-critical state, passing it through has little influence on how things end.

Reply with exactly one of:
<judge>yes</judge> -- high criticality: intervening on this handoff strongly influences the final outcome
```
```

```
<judge>no</judge> -- low criticality: this handoff can pass as-is with little
influence on the final outcome
```

Multi-agent slow-path prompt

```
You are a steering supervisor watching over a multi-agent system that is solving
a task. The system is a fixed pipeline of role agents. One agent has just
finished, and its output is about to be handed to the downstream agent(s).

Task:
```txt
{task}
```

Pipeline:
```txt
{pipeline}
```

{history_block}

Handoff under review:
- from: {from_desc}
- to: {to_desc}
- position: {position}
- submission:
```txt
{submission}
```

Retries used at this handoff: {retries_used} of {max_retries}.{
retry_feedback_block}

You have decided to intervene at this handoff. Think through what is wrong with
the submission or what the receiving agent needs to know, then commit exactly one
 action:

<retry>your rejection feedback -- the submission is rejected and the rejection is
 returned to the submitting agent as the result of its submission; it will redo
the work with your feedback in view</retry>
<note>your note -- the submission passes through unchanged, and this note is
attached to it for the receiving agent(s)</note>
```

Planning prompt

```
You are a steering supervisor watching over an autonomous search agent. The agent
 solves a question with a search engine; you see exactly what the agent sees: the
 question and its action history so far. The agent is about to decide its next
action.

Question:
```txt
{question}
```

Agent's action history:
```txt
{history}
```
{prior_notes_block}
```

```
Agent status: it has taken {num_actions} actions; at most {remaining_turns} turns
 remain before it must answer.
{action_space}

The agent has not acted yet. Think through how it should approach this question,
then write a steering note that the agent will keep in view for the whole task.

After your reasoning, reply with the note inside tags:
<note>your steering note (1-3 concise sentences: point out the problem and what
to do next)</note>
```

Verification prompt

```
You are a steering supervisor watching over an autonomous search agent. The agent
 solves a question with a search engine; you see the question and its full action
 history, exactly as the agent sees them. The agent has just submitted an answer;
 you decide whether it stands.

Question:
```txt
{question}
```

Agent's action history:
```txt
{history}
```

Agent status: it has taken {num_actions} actions; at most {remaining_turns} turns
 remain before it must answer.
{action_space}

The agent has submitted the answer shown as the last turn of its history. Decide
whether to accept it. Think it through, then commit exactly one action:

<pass></pass> -- the answer is delivered unchanged, no intervention
<retry>your rejection feedback -- the submission is rejected and the rejection is
 returned to the agent as the result of its submission; it will redo the work
with your feedback in view</retry>
```

LLM-as-Judge prompt

```
You are an evaluation assistant. Please determine if the predicted answer is
equivalent to the labeled answer.

Question: {question}

Labeled Answer: {gt_answer}

Predicted Answer: {pred_answer}

Did the model give an answer **equivalent** to the labeled answer? Please respond
 with "Correct" if they are equivalent, or "Incorrect" if they are not equivalent
.

The output should in the following json format:
```json
{
    "rationale": your rationale for the judgement, as a text,
    "judgement": your judgement result, can only be "Correct" or "Incorrect",
}
```
```

```
```