

Deliberation on Demand: Selective Deep Thinking for Language Agents

Junnian Chen¹ Jianxun Lian^{2,*} Qinlu Cao¹ Chunlin Liu¹

Jiwei Zhang¹ Mingyang Zhou¹ Hao Liao^{1,*}

¹College of Computer Science and Software Engineering, Shenzhen University

²Microsoft Research Asia

chenjunnian2023@email.szu.edu.cn

jianxun.lian@microsoft.com, haoliao@szu.edu.cn

*Corresponding authors

Abstract

Large reasoning models improve difficult problem solving by spending many tokens on internal reasoning. This paradigm is natural for single-turn tasks, but poorly matched to language agents: an agent may take tens or hundreds of environment steps, making deep thinking before every action costly and sometimes harmful. We study *agentic deep thinking*: when should an agent invoke long-form private deliberation, and how should it be exposed inside a ReAct-style loop? We propose a selective deep-thinking framework that separates the short externalized Thought used to choose the next action from a longer private memo used to reorganize future behavior. Across three benchmarks and six model families, selective deep thinking improves agents, but the best interface depends on backbone capability: stronger models benefit most from deliberation as a callable action, whereas smaller models are more robust with a separate gate. We further show that RL can teach a less capable backbone when to deliberate and how to act on the resulting memo, improving both task performance and transfer while reducing unnecessary reasoning. These results suggest that deep thinking is most useful for language agents when it is selective, controllable, and matched to model capability.

1 Introduction

Recent large reasoning models, such as OpenAI’s o-series models and DeepSeek-R1, show that additional internal reasoning can improve hard problem solving (OpenAI et al., 2024; DeepSeek-AI et al., 2025). For one-shot tasks such as complex math, this trade-off is natural: one long reasoning trace leads to one answer. Language agents operate in a different regime. A ReAct-style agent (Yao et al., 2023b) repeatedly observes, reasons, acts,

and updates context across embodied tasks (Shridhar et al., 2021), multi-hop tool use (Ye et al., 2025; Qin et al., 2024), and open-domain assistant tasks (Mialon et al., 2024). If every step launches a full deep-thinking process, cost scales with trajectory length and deliberation can crowd out action.

This creates a central tension. ReAct’s short Thought field is a useful speak-aloud rationale for the next action, but it is not a full planning episode. Deep thinking plays a different role: it is a private deliberation process for summarizing progress, re-considering assumptions, detecting dead ends, and forming multi-step plans. Agents need this second mode, but not at every step. Thus the key question is not only *how much* a model should think, but *when* an agent should allocate deep thinking during an interactive trajectory. Prior work on test-time reasoning budgets often studies single-turn settings and asks when not to think (Chen et al., 2025; Ma et al., 2025; Zhang et al., 2025a). In an agent loop, deliberation at step t can reshape later context, actions, and observations, making it an intervention in a sequential decision process rather than a longer pre-answer trace.

Our approach. We propose a selective deep-thinking framework for language agents. The framework separates two reasoning channels in a shared ReAct backbone: an *externalized action rationale* r_t , which is the short Thought used to choose the next environment action, and a *private deliberation memo* z , which is produced by a longer deep-thinking pass and appended to future context. The central design problem is the interface that controls when this memo is produced. We instantiate *Periodic Thinking* (fixed schedule), *Sel.-Gate* (a binary `deep_think` decision before each action), and *Sel.-Action* (`deep_think` as a callable tool).

We first evaluate the framework without model post-training, isolating what can be gained from

Code is available at [GitHub](#).

the interface alone. Selective deep thinking helps, but the best interface depends on backbone capability. Strong proprietary models often use Sel.-Action effectively, while smaller open-source backbones tend to under-use or misplace deep thinking.

This capability dependence motivates the second part of the paper. If a backbone does not naturally know how to use the selective interface, we can learn that interface competence, which involves both deciding when to deliberate and adapting the downstream policy to act on the resulting memo. We train the selective policy with GRPO using task success and efficiency as rewards, so the model must discover when deep thinking is useful from complete trajectories rather than from an explicit thinking bonus. We evaluate these designs on three agentic task families: embodied household instruction following, multi-hop tool use, and open-domain assistant-style question answering. Across six proprietary and open-source model families, we compare standard ReAct, fixed-schedule deliberation, selective gating, selective action, and RL-trained selection to measure both task success and token efficiency.

Major contributions of this paper are:

- **A selective deep-thinking framework for agents.** We propose a ReAct-compatible framework that distinguishes short externalized action rationales from long private deliberation memos, making deep thinking a controllable operation inside the agent loop.
- **Learning to use the deliberation interface.** We train the selective policy with GRPO and show that reinforcement learning can teach a less capable backbone both when to deliberate and how to act on the resulting memo, reversing the zero-shot interface ranking and transferring to unseen benchmarks.
- **Substantial agentic-task experiments.** We evaluate across three agentic tasks and six model families, comparing Periodic Thinking, Sel.-Gate, and Sel.-Action across embodied decision-making, multi-hop tool use, and open-domain assistant settings.

2 Related Work

2.1 Deliberation in Language Agents

ReAct (Yao et al., 2023b) establishes a standard tool-using agent loop: a chain-of-thought Thought

turn (Wei et al., 2022) selects an action, and the resulting observation is fed into the next step. Later work adds trajectory-level correction (Shinn et al., 2023; Madaan et al., 2023), richer action spaces (Schick et al., 2023; Qin et al., 2024; Wang et al., 2024b; Xu et al., 2023), and planning/memory modules (Wang et al., 2023a; Yao et al., 2023a; Besta et al., 2024; Wang et al., 2024a; Packer et al., 2023; Park et al., 2023; Li et al., 2026). They improve what agents remember, plan, or repair, but usually do not make long-form deliberation a controllable operation inside each ReAct turn.

Closest to our setting, PReAct (Rawat et al., 2025) keeps a structured execution plan available at every step, LWTP (Paglieri et al., 2025) learns when to emit a planning token, and Zhu et al. (2025) show that selective step-level reflection beats always-reflecting in agents. DPT-Agent (Zhang et al., 2025b) also introduces a System-2 module, but in an asynchronous human-AI collaboration setting (He et al., 2024). We share the motivation that agents need more than local action rationales, but focus on *inline* private deliberation for single-agent task solving: three interface designs, their dependence on backbone capability, and whether RL can teach a weaker model to use the interface.

Our memo is an exploratory, non-binding reasoning aid rather than an executable plan; planning commits an artifact for execution, whereas learned memory managers (Yan et al., 2026; Yu et al., 2026) optimize what to store, update, retrieve, or delete. These directions are complementary.

2.2 Adaptive and Learned Deliberation

Reasoning models such as o1 (OpenAI et al., 2024), DeepSeek-R1 (DeepSeek-AI et al., 2025), and Qwen QwQ (Qwen Team, 2024) show that extended reasoning (Zelikman et al., 2022) and scaled test-time compute (Snell et al., 2025; Muenighoff et al., 2025) can improve difficult problem solving; self-consistency (Wang et al., 2023b) improves answers through aggregation. Other work asks when not to think: AdaptThink (Zhang et al., 2025a) adapts reasoning at the token level, while no-thinking and overthinking studies show that long reasoning can be unnecessary on easier inputs (Ma et al., 2025; Chen et al., 2025).

These works mainly treat deliberation as a single-turn budget decision. We study the agentic version: a deep-thinking call at step t changes

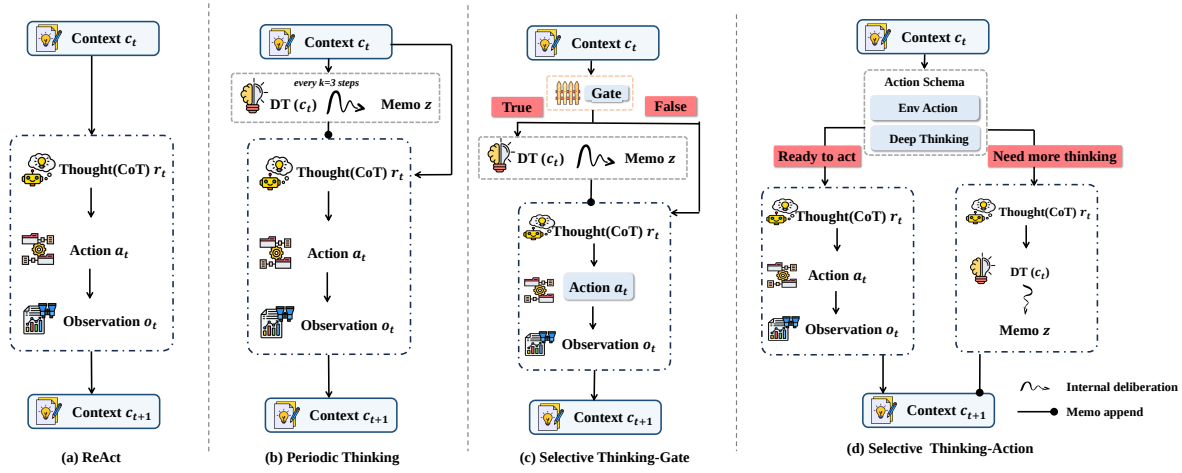


Figure 1: Deliberation interfaces in a shared ReAct backbone. The externalized action rationale r_t chooses actions, while optional private deliberation $DT(c_t)$ produces memo z appended to context to guide future turns. **(a) ReAct:** action rationale only. **(b) Periodic Thinking:** fixed schedule, DT every $k = 3$ steps. **(c) Sel.-Gate:** lightweight binary decision before each turn. **(d) Sel.-Action:** deliberation as a callable action; GRPO learns when to invoke `deep_think`.

the context and action distribution for later steps. Our GRPO training (Shao et al., 2024) therefore learns the deliberation-control decision from task-completion rewards, complementing broader RLHF (Ouyang et al., 2022) and agent-RL approaches (Zhou et al., 2024; Rafailov et al., 2023).

3 Method

Our method has two stages. First, we introduce a selective deep-thinking framework that can be used at inference time with an off-the-shelf backbone: the agent keeps the standard ReAct action loop, but gains a controlled deliberation channel that can be invoked only when useful. This stage asks whether agentic deep thinking helps without any model post-training, and which interface lets different backbones use it effectively. Second, motivated by the fact that weaker open-source backbones may not reliably master this new interface from prompting alone, we study whether reinforcement learning can teach the model *when* to deliberate and how to act after deliberating.

3.1 Background: the ReAct Loop

We model an agent episode as interaction with a partially observable environment. Let x denote the task instruction and let $o_{<t} = (o_1, \dots, o_{t-1})$ be the observations produced by previous environment actions. The agent’s textual context before

step t is

$$c_t = (x, r_1, a_1, o_1, \dots, r_{t-1}, a_{t-1}, o_{t-1}, m_t), \quad (1)$$

where m_t is an optional deliberation memo, defined below. A standard ReAct policy generates an externalized action rationale r_t followed by an environment action a_t :

$$\begin{aligned} r_t &\sim \pi_{\theta}^R(\cdot \mid c_t); & a_t &\sim \pi_{\theta}^A(\cdot \mid c_t, r_t); \\ o_t &\sim \mathcal{E}(\cdot \mid h_t, a_t), \end{aligned} \quad (2)$$

where h_t is the latent environment state. Equivalently, $\pi_{\text{ReAct}}(r_t, a_t \mid c_t)$ factorizes into the two textual generations above. This notation separates two roles that are often collapsed under the name “chain-of-thought”: r_t is the public Thought field that explains and conditions the immediate action, whereas the environment transition depends only on a_t .

The key limitation is temporal granularity. In long-horizon tasks, an agent may take tens or hundreds of turns. Running an expensive long-form reasoning process before every action is therefore wasteful, yet relying only on the short ReAct Thought often leaves the agent without a mechanism for summarizing progress, reconsidering failed hypotheses, or planning several turns ahead. Our method treats such long-form reasoning as a controllable operation inside the agent loop.

3.2 Two Reasoning Channels: Action Rationale and Private Deliberation

We distinguish two reasoning channels with different visibility, duration, and computational roles. *Externalized action rationale* denotes the ReAct Thought r_t : a concise, speak-aloud justification attached to a single action. *Private deliberation* denotes a longer internal reasoning episode that is invoked only when needed and whose output is compressed into a persistent memo. We use *deep thinking* for this second channel and write

$$z_t = \text{DT}_\theta(c_t), \quad (3)$$

where DT_θ is implemented by the same backbone LLM under a deliberation prompt. Let $d_t \in \{0, 1\}$ indicate whether deep thinking is invoked before the step- t action (the controller that produces d_t is specified in Eq. 5). The memo state is then updated as

$$m_{t+1} = \begin{cases} z_t, & d_t = 1, \\ m_t, & \text{otherwise.} \end{cases} \quad (4)$$

Externalized action rationale r_t . The rationale r_t corresponds to the *Thought* turn in the *Thought* $\rightarrow$ *Action* $\rightarrow$ *Observation* cycle (Yao et al., 2023b). It is generated online from c_t , normally short in our traces, and immediately followed by a_t . Its function is local: to state the currently relevant observation, the immediate sub-goal, and why the selected tool or environment action is appropriate. Although past rationales remain in the context window and can support episode coherence, each new r_t is tied to the next action decision rather than to a separate planning process.

Private deliberation memo z_t . Deep thinking is a side-effect-free internal operation. It does not act on the environment and does not receive a new environment observation. Instead, it allocates a longer generation to the agent’s latent deliberation work: consolidating progress, revisiting assumptions, recognizing dead ends, exploring alternative sub-goals, and deciding what should be tried next. The output is not treated as another action rationale. It is a structured memo z_t that is appended to future contexts as m_{t+1} , so subsequent rationales and actions can condition on it until a later memo replaces it. This design reflects the distinction between a brief “speak-aloud” explanation and a

longer private reasoning process: the former accompanies action, while the latter reorganizes the agent’s state of mind for future turns.

Deliberation controller. The remaining design question is when to pay for DT_θ . Recall that $d_t \in \{0, 1\}$ (introduced in Eq. 4) indicates whether private deliberation is invoked before the step- t action. A deliberation interface specifies the controller

$$d_t \sim q_\phi(\cdot \mid c_t), \quad (5)$$

and the agent then samples (r_t, a_t) from the ReAct policy conditioned on the updated memo. Different choices of q_ϕ give the interfaces we evaluate. The token `deep_think` is the concrete API symbol: it appears either as a callable action in Sel.-Action or as a binary field in Sel.-Gate.

Periodic Thinking. Periodic Thinking uses a hand-coded controller, $d_t = \mathbb{I}[t \equiv 0 \pmod k]$, and runs DT_θ before the next ReAct turn whenever the controller fires. We use $k = 3$ throughout. This baseline asks whether a simple regular allocation of private deliberation is already enough to improve long-horizon agents.

Selective Thinking as Action (Sel.-Action). Sel.-Action exposes `deep_think` as an additional tool in the same action schema as the environment tools. The policy first emits a rationale and then selects either a real environment action or `deep_think`:

$$a_t \in \mathcal{A}_{\text{env}} \cup \{\text{deep_think}\}. \quad (6)$$

If `deep_think` is selected, no environment transition occurs; instead, $\text{DT}_\theta(c_t)$ is run and the resulting memo is returned to the agent as the tool output. To prevent all-deliberation trajectories, `deep_think` is removed from the schema for the immediately following turn and restored on the next one. This interface gives the agent maximal autonomy: deciding whether to deliberate and which environment action to take are both represented inside the same action distribution.

Selective Thinking Gate (Sel.-Gate). Sel.-Gate separates the deliberation decision from the action decision. Before each ReAct turn, the model emits a constrained binary field, `deep_think: true | false`. When the field is true, we run $\text{DT}_\theta(c_t)$, update the memo, and then generate the usual (r_t, a_t) pair. The gate prompt is asymmetric and default-false: it lists narrow triggers such

as unexplained observations, repeated failed attempts, or an impending multi-step plan that has not been verified. This structure turns “whether to think” into a lightweight classification problem, which can be easier for smaller models than choosing `deep_think` from a full tool schema.

For reinforcement learning (§3.3), the controller tokens are part of the optimized trajectory: in Sel.-Action these are the tokens that select `deep_think`, while in Sel.-Gate they are the binary gate tokens. Whenever deliberation is invoked, the model regenerates the memo and replaces its previous version in the context. GRPO does not directly optimize these generated memo tokens; its learning signal flows through the deliberation decision and subsequent policy actions.

3.3 Learning Selective Deliberation

The previous subsection treats selective deep thinking as an *inference-time* framework: the backbone is prompted with a deliberation interface, but its parameters are unchanged. This is important because it isolates the value of the framework itself. However, it also exposes a capability dependence. Stronger models can often treat `deep_think` as a useful high-level action, whereas smaller open-source backbones may under-use it, call it in trivial states, or fail to translate the resulting memo into better subsequent actions. We therefore introduce a second stage: learn the deliberation policy directly from task outcomes, so that a less capable backbone can acquire the interface competence that prompting alone may not elicit.

Concretely, we ask whether either selective interface can be *learned* to outperform its zero-shot counterpart. We frame each rollout as a trajectory τ producing a binary task outcome $won \in \{0, 1\}$ and we optimise the policy with Group Relative Policy Optimization (GRPO) (Shao et al., 2024), which estimates trajectory-level advantages from sampled groups and avoids fitting a separate value network (Schulman et al., 2017). We train two selective policies under the same reward, RL-Action and RL-Gate, reporting RL-Action as our primary policy. We also train two matched attribution controls, RL-ReAct and RL-Gate-only:

RL-Action. The full ReAct policy is fine-tuned end-to-end while `deep_think` is exposed as a callable tool in the action schema alongside the environment tools (Figure 1d). The credit assigned

by GRPO therefore flows through *both* the deliberation choice and the subsequent environment action, so the model can co-adapt *when* to think and *what* to do next.

RL-Gate. RL-Gate fine-tunes the two-stage gate interface defined in §3.2: GRPO updates both the binary gate and the downstream Reason+Action policy, but the gate decision and environment action remain separately generated fields rather than a unified action space. Because the gate affects outcomes only through the subsequent policy, credit assignment is less direct than in RL-Action (§4.2).

For attribution, RL-Gate-only isolates controller learning by using the same gate and reward while freezing the downstream Reason+Action policy; RL-ReAct isolates generic policy optimization by removing the separate `deep_think` channel under the same objective and training budget. We train all four variants on Qwen3-8B using ALFWorld and compare their inference-time interface robustness and RL learnability in §4.2.

Reward design. We avoid an explicit positive reward for deep thinking so that GRPO must *discover* its value through outcome differences within a group. The reward factorises as a task signal multiplied by a compound efficiency discount:

$$R(\tau) = R_{\text{succ}}(\tau) (1 - C_{\text{tok}}(\tau) - C_{\text{dt}}(\tau)), \quad (7)$$

with three components defined as

$$R_{\text{succ}}(\tau) = \mathbb{I}_{\text{succ}}(\tau), \quad (8)$$

$$C_{\text{tok}}(\tau) = \lambda_{\text{tok}}(s_t) \frac{L(\tau)}{L_{\text{max}}}, \quad (9)$$

$$C_{\text{dt}}(\tau) = \lambda_{\text{dt}} [N_{\text{dt}}(\tau) - \bar{N}_{\text{dt}}(t) - 1]_+. \quad (10)$$

Here $[x]_+ = \max(0, x)$; $L(\tau)$ is the total output token count (*including* DT tokens) and $L_{\text{max}} = 4096$ is the episode budget; $N_{\text{dt}}(\tau)$ is the number of deep-thinking calls; $\bar{N}_{\text{dt}}(t)$ is an adaptive reference updated online as the moving average of N_{dt} over successful rollouts in a 128-episode window (initialised at 5, the empirical sweet spot for ALFWorld); and $\lambda_{\text{dt}} = 0.01$ is a fixed small coefficient.

Capability-gated token coefficient. $\lambda_{\text{tok}}(s_t)$ is gated by the moving-average success rate $\bar{\text{SR}}$:

$$\lambda_{\text{tok}}(s_t) = \begin{cases} 0 & \bar{\text{SR}} < 0.60, \\ 0.05 & \bar{\text{SR}} \geq 0.60. \end{cases}$$

We use a hard switch rather than a smooth ramp; the 128-episode success-rate moving window already provides ample temporal smoothing, and our ablations showed that linear ramps in the sub-threshold regime contributed values too small ($\lambda_{\text{tok}} < 0.02$) to produce a usable gradient signal in practice. Phase 1 ($\text{SR} < 0.60$) keeps token pressure off the policy while it is still learning to solve the task; Phase 2 activates C_{tok} once the agent reliably succeeds on the majority of episodes.

Design rationale. Three properties make the reward GRPO-friendly. (i) The only positive term is R_{succ} , so within-group advantages naturally flow to trajectories that solve the task. We deliberately avoid any explicit bonus for deep thinking: such a bonus collapses a degenerate failure mode we observed during early prototyping—without success gating, C_{tok} teaches GRPO that “solved with $N_{\text{dt}} = 0$ ” strictly dominates “solved with $N_{\text{dt}} > 0$ ”, and deliberation vanishes within a few thousand steps. (ii) Both C_{tok} and C_{dt} are multiplied by R_{succ} : failed episodes already carry the maximal negative signal (zero task reward), so we do not compound that with DT penalties, preserving exploration in early training when most episodes still fail. (iii) The adaptive reference $\bar{N}_{\text{dt}}(t)$ creates a *width-1 neutral zone*—deviations $N_{\text{dt}} \in \{\bar{N}_{\text{dt}} - 1, \bar{N}_{\text{dt}} + 1\}$ incur no penalty, letting GRPO compare them through success rate alone and letting $\bar{N}_{\text{dt}}$ drift to the empirically efficient frontier rather than locking onto a hard-coded target. A quantitative magnitude analysis of the three reward components (and why each remains an order of magnitude below the task signal) is provided in Appendix B.

4 Experiments

Benchmarks. We evaluate on three realistic agent benchmarks chosen to cover diverse practical task settings: **ALFWorld** (Shridhar et al., 2021) provides long-horizon, partially observable embodied tasks in TextWorld (134 unseen episodes), where state-tracking and sub-goal management across many steps dominate. **ToolHop** (Ye et al., 2025) contains 995 multi-hop queries over 3,912 tools requiring feedback-conditioned sequential tool invocation; we report success rate, output tokens, and a process-quality score measuring intermediate tool-call correctness. **GAIA** (Mialon et al., 2024) tests open-domain assistant behaviour through 103 text-only public valida-

tion items combining web search, document verification, and multi-step synthesis. We use the `smolagents` harness with a maximum of 25 agent steps per item; details are given in Appendix A. Together, the three benchmarks span embodied state-tracking, tool-chain decomposition, and open-domain search, giving a broad view of how the interfaces behave across distinct real-world agent workflows.

Backbones. Six models spanning open-source and proprietary, small and large: Qwen3-8B¹, Qwen3-32B (Yang et al., 2025), Llama-3.3-70B-Instruct (Grattafiori et al., 2024), GPT-4o (OpenAI et al., 2023), DeepSeek-V4-Pro Preview (DeepSeek-AI, 2026), and GPT-5.2.

Modes. Per backbone we run six core inference modes: **ReAct** (Yao et al., 2023b), **Reflexion** (Shinn et al., 2023), **Always Thinking**, **Periodic Thinking** ($k = 3$), **Sel-Action**, and **Sel-Gate**. We additionally benchmark two external planning baselines on all six backbones: **Dyn-Replan** (Paglieri et al., 2025), which regenerates a high-level plan whenever the agent detects its current plan has become invalid, and **Pre-Act** (Rawat et al., 2025), which maintains an incrementally updated multi-step execution plan in context at every step. All modes use the same backbone, tools, benchmark instances, and step budget; method-specific planning or deliberation instructions differ as required by each baseline.

Implementation. Decoding uses the model’s default sampling settings; experiments are seed-matched across modes. For RL we train Qwen3-8B with GRPO on the ALFWorld training split (group size 8, 128-episode success-rate window) and evaluate the learned policy zero-shot on all three test sets. Training details and hyperparameters are in Appendix A.

4.1 Main Results

Table 1 summarizes the main results; results for all backbones are in Appendix H. Across the four backbones shown in Table 1, the best selective interface exceeds both planning baselines in mean success rate. Selective interfaces also avoid the large token growth of Always Thinking in most settings, although savings relative to ReAct vary by backbone and benchmark.

¹<https://huggingface.co/Qwen/Qwen3-8B>

Model	Mode	ALFWorld		ToolHop		GAIA		Mean	Success	Output Token
		Success Rate	Output Tokens	Success Rate	Output Tokens	Success Rate	Output Tokens	Success Rate	Rate vs. ReAct	Change vs. ReAct (%)
GPT-5.2	ReAct	78.4	2.0k	55.2	727	41.7	16.8k	58.4	—	—
	Dyn.-Replan	75.4	1.2k	63.8	646	36.9	3.0k	58.7	+0.3	-75.2%
	Pre-Act	70.1	1.7k	65.8	681	36.9	3.5k	57.6	-0.8	-69.9%
	Reflexion	77.6	2.2k	57.7	885	41.7	16.1k	59.0	+0.6	-1.8%
	Always	84.3	4.1k	66.4	2.2k	<u>44.7</u>	34.6k	65.1	+6.7	+109.5%
	Periodic (ours)	89.6	2.5k	62.1	878	46.6	18.5k	66.1	+7.7	+12.0%
	Sel.-Gate (ours)	82.8	1.1k	58.1	795	45.6	20.3k	62.2	+3.8	+13.7%
	Sel.-Action (ours)	91.0	1.5k	<u>65.1</u>	760	46.6	16.9k	67.6	+9.2	-1.9%
DeepSeek-V4-Pro Preview	ReAct	73.1	2.0k	65.1	615	30.1	4.3k	56.1	—	—
	Dyn.-Replan	82.8	2.2k	68.9	914	35.9	4.5k	62.5	+6.4	+10.1%
	Pre-Act	76.1	2.4k	68.4	914	32.0	5.4k	58.8	+2.7	+26.0%
	Reflexion	70.9	1.2k	68.5	621	34.0	6.6k	57.8	+1.7	+21.8%
	Always	79.9	2.6k	69.1	2.0k	35.0	8.9k	61.3	+5.2	+95.2%
	Periodic (ours)	78.4	1.3k	67.8	883	36.9	4.8k	61.0	+4.9	+1.0%
	Sel.-Gate (ours)	79.9	1.7k	62.4	1.5k	34.0	7.1k	58.8	+2.7	+49.0%
	Sel.-Action (ours)	83.6	1.3k	69.8	1.0k	<u>35.9</u>	4.0k	63.1	+7.0	-8.9%
Llama-3.3-70B-Instruct	ReAct	49.3	5.3k	54.4	537	13.6	0.9k	39.1	—	—
	Dyn.-Replan	49.3	4.2k	55.7	697	15.1	1.3k	40.0	+0.9	-8.0%
	Pre-Act	38.8	2.5k	58.4	1.0k	16.0	1.0k	37.7	-1.4	-33.2%
	Reflexion	49.3	5.5k	54.9	678	21.4	1.2k	41.9	+2.8	+9.5%
	Always	53.0	10.3k	53.4	1.8k	21.4	2.9k	42.6	+3.5	+122.7%
	Periodic (ours)	64.2	6.3k	57.2	1.2k	22.3	1.6k	47.9	+8.8	+35.1%
	Sel.-Gate (ours)	60.4	4.9k	<u>57.7</u>	926	22.3	1.2k	46.8	+7.7	+4.3%
	Sel.-Action (ours)	50.0	4.7k	53.2	932	18.4	1.3k	40.5	+1.4	+2.9%
Qwen3-8B	ReAct	46.3	3.1k	39.6	628	12.6	3.7k	32.8	—	—
	Dyn.-Replan	50.7	2.4k	43.1	1.1k	12.8	4.2k	35.5	+2.7	+3.7%
	Pre-Act	44.0	3.2k	44.1	0.8k	8.8	3.8k	32.3	-0.5	+17.7%
	Reflexion	45.5	3.4k	35.7	540	15.5	4.8k	32.2	-0.6	+5.0%
	Always	49.3	5.2k	43.5	1.2k	14.6	7.1k	35.8	+3.0	+81.7%
	Periodic (ours)	53.0	3.5k	44.3	984	16.5	6.5k	<u>37.9</u>	+5.1	+47.9%
	Sel.-Gate (ours)	52.2	3.0k	45.1	1.0k	18.4	3.4k	38.6	+5.8	-0.4%
	Sel.-Action (ours)	45.5	3.3k	42.6	793	14.6	3.9k	34.2	+1.4	+7.6%
Qwen3-8B-RL	ReAct (control)	55.2	1.5k	29.7	491	9.7	2.6k	31.5	-1.3	-38.2%
Qwen3-8B-RL	Gate-only (control)	54.5	2.9k	45.6	1.1k	17.4	3.6k	39.2	+6.4	+2.3%
Qwen3-8B-RL	Sel.-Gate (ours)	61.9	2.7k	47.7	989	20.4	3.3k	43.3	+10.5	-5.9%
Qwen3-8B-RL	Sel.-Action (ours)	67.9	<u>2.3k</u>	48.2	<u>764</u>	21.4	4.2k	45.8	+13.0	-2.2%

Table 1: Main results for the four selected backbones and the Qwen3-8B-RL attribution controls. Bold and underlined values are the best and second-best within each backbone. Light blue marks our inference-time interfaces. Token Change is the percentage change in mean output tokens relative to the corresponding ReAct baseline; positive values indicate an increase and negative values indicate a reduction. All percentages are rounded to one decimal place.

The best interface depends on model capability.

The central pattern is that Sel.-Action is strongest on advanced proprietary backbones like GPT-5.2, but not on smaller open-source backbones. This is expected: Sel.-Action asks the model to decide, inside the same tool-invocation style used for environment actions, whether `deep_think` should be called. Stronger models can treat this as a fluent meta-action and schedule deliberation only when it helps; weaker base models often fail to use the action interface consistently enough for the additional control to pay off.

Selective thinking remains useful under weaker backbones. For open-source backbones, the

safer inference-time choices are Sel.-Gate and Periodic Thinking. These interfaces reduce the burden on the backbone: Periodic Thinking supplies an external schedule, while Sel.-Gate isolates the deliberation decision from the ordinary action grammar. As a result, open-source models still benefit from selective deliberation, but the best interface shifts away from Sel.-Action until the model is trained to understand the new mechanism.

4.2 Learning to Schedule Deliberation

RL unlocks the action interface. We post-train Qwen3-8B under both selective interfaces using only the ALFWorld training split, then directly

evaluate the resulting policies on ALFWorld, ToolHop, and GAIA. Both RL models outperform all inference-time Qwen3-8B settings in Table 1, from ReAct through Sel.-Action. This indicates that the base model’s weakness lies in how it *utilizes* deliberation rather than in a lack of useful deliberation content: as §C.2 shows, its deliberation often diagnoses the problem correctly while the following action still ignores it. Utilization has two components, and the gains arise from their joint effect: learning *when* to deliberate, and co-adapting the downstream policy to act on the resulting memo. The RL-Gate-only ablation (§B) confirms that learning the controller alone, with the downstream policy frozen, recovers only a small part of the improvement on ALFWorld. **Because the RL policies are trained only on ALFWorld but transfer directly to ToolHop and GAIA after just 150 training steps, the results suggest transfer beyond the ALFWorld training environment.** After training, Sel.-Action is clearly better than Sel.-Gate, echoing our hypothesis that the action interface is more expressive once the backbone learns to use it as part of the agent’s tool-invocation policy.

RL reshapes deliberation strategy. Figure 2 compares the per-step trigger frequency before

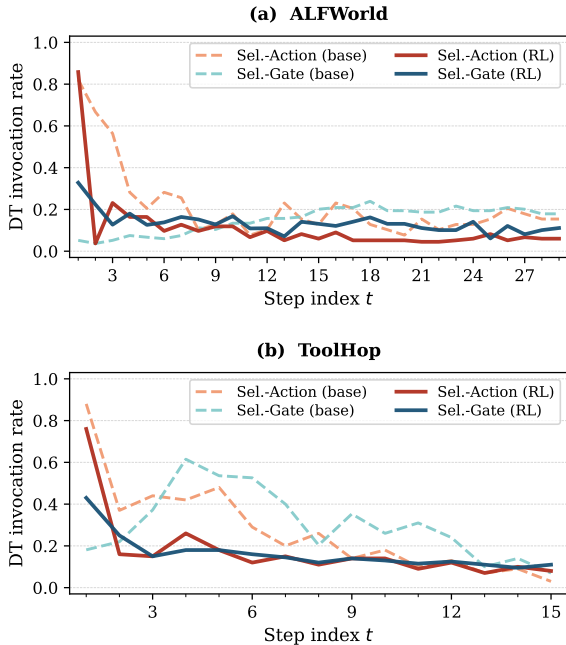


Figure 2: Per-step deep-think trigger frequency for Qwen3-8B on (a) ALFWorld and (b) ToolHop. The y -axis is the probability of invoking the Thinking action or gate at step k . Each point averages over 134 (ALFWorld) and 995 (ToolHop) evaluation tasks.

and after RL training on ALFWorld and ToolHop. Across both benchmarks, the RL-trained Sel.-Action and Sel.-Gate policies show a consistent plan-early, act-then-adapt rhythm: they trigger deep thinking at the beginning of the trajectory and then become sparse, with only occasional re-evaluation later. The base Sel.-Action policy shows a consistent failure mode in the opposite direction, over-triggering at the beginning on both benchmarks before decaying. The base Sel.-Gate policy is more benchmark-dependent: on ALFWorld it starts with a low trigger rate and gradually rises as the trajectory unfolds, while on ToolHop it rises sharply after the first few steps, stays high through the early-to-middle tool chain, and only then tapers off. These patterns suggest that RL learns a more stable temporal scheduling strategy across tasks, rather than merely shifting the overall trigger rate.

4.3 Case Studies

Deep thinking as missing meta-cognition (Figure 3). ReAct activates the desk lamp at step 2 then loops on `look/examine desk 1` for 47 steps, re-generating the same Thought (“*lamp is on; cd must be here*”) without ever visiting `desk 2`. Sel.-Action prevents this by investing one DT call at step 1, decomposing the task into two independent sub-goals (*locate desk lamp* and *locate cd*) before any search begins. Sel.-Gate needs no deliberation for steps 1–5; it fires only at step 6—after picking up the `cd` at `desk 2`—to note the unresolved spatial precondition and steer the agent back to the lamp. Both solve the task in 6 steps; their difference is purely temporal: proactive decomposition versus reactive correction at the last sub-goal transition.

4.4 RL Attribution and Ablations

The four RL rows in Table 1 isolate what is learned by reinforcement learning on Qwen3-8B. RL-ReAct removes the deliberation channel while retaining the same training objective, whereas RL-Gate-only trains the gate but freezes the downstream ReAct policy. The two full policies update both the controller and the action policy. RL-Action achieves the strongest mean success rate (45.8) and the largest gain over ReAct (+13.0), with RL-Gate second (43.3, +10.5). RL-Gate-only reaches 39.2 (+6.3), and RL-ReAct reaches 31.5 (−1.3), showing that neither controller learning nor generic RL adaptation alone fully explains

Task: "examine the cd with the desklamp."		
ReAct ✗	Sel.-Gate ✓	Sel.-Action ✓
Obs 1: Room: desk1, desk2, drawers, shelves, bed... Th 1: Need to find desklamp first; desks are the most likely location. Act 1: go to desk 1 Obs 2: desk1: desklamp1, pencil1 Th 2: Desklamp is here; activate it, then locate the cd on desk1. Act 2: use desklamp 1 Obs 3: You turn on the desklamp1. Th 3: Lamp is on; the cd should be visible on desk1 under the lamp. Act 3: examine desk 1 Obs 4: desk1: desklamp1, pencil1 Th 4: The desklamp is on; cd must be here--keep looking around desk1. Act 4: look Obs 5: You are facing desk1. Next to it, you see nothing. Th 5: The desklamp is on and cd must be here--look once more. Act 5: look ... Steps 6-49: identical ... Obs 50: You are facing desk1. Next to it, you see nothing. Th 50: The desklamp is on; I should keep checking--cd must be here somewhere. Act 50: look Steps 4-50: loops between look, examine desk1, and use desklamp1--never moves to desk2. ✗ MAX STEPS (50). FAIL.	Obs 1: Room: desk1, desk2, drawers, shelves... Gate 1: False→skip Th 1: Check desks for desklamp first. Act 1: go to desk 1 Obs 2: desk1: desklamp1, pencil1 Gate 2: False→skip Th 2: Desklamp found; activate it then locate cd. Act 2: use desklamp 1 Obs 3: You turn on the desklamp1. Gate 3: False→skip Th 3: Lamp on; check desk1 for cd. Act 3: examine desk 1 Obs 4: desk1: desklamp1, pencil1 Gate 4: False→skip Th 4: No cd on desk1; move to desk2 next. Act 4: go to desk 2 Obs 5: desk2: bowl1, cd3, mug1 Gate 5: False→skip Th 5: cd found; take it, then return to the lamp to examine. Act 5: take cd3 from desk2 Obs 6: You pick up the cd3. Gate 6: True→DT <i>Holding cd at desk2; desklamp is at desk1--return to lamp to finish.</i> Th 6: Back to desklamp on desk1 to examine the cd under it. Act 6: go to desk 1 ✓	Obs 1: Room: desk1, desk2, drawers, shelves... Th 1': Locate desklamp and cd, likely on desks or shelves; plan search order. Act 1': deep_think <i>Missing precondition: locate both desklamp and cd. Check desktops first--lamp most likely on desk, cd too. Best next: go to desk1.</i> Th 1: Locate desklamp and cd; desks are the most likely place to find them. Act 1: go to desk 1 Obs 2: desk1: desklamp1, pencil1 Th 2: Desklamp found; activate it. Act 2: use desklamp 1 Obs 3: You turn on the desklamp1. Th 3: Lamp on; check desk1 for cd. Act 3: examine desk 1 Obs 4: desk1: desklamp1, pencil1 Th 4: No cd here; go to desk2 as planned. Act 4: go to desk 2 Obs 5: desk2: bowl1, cd3, mug1 Th 5: cd3 found; take it and return to the desklamp. Act 5: take cd3 from desk2 Obs 6: You pick up the cd3. Th 6: Holding cd; return to lit lamp at desk1 to complete the task. Act 6: go to desk 1 ✓

Figure 3: Case study: "examine the cd with the desklamp" (ALFWorld, trial idx = 20, DeepSeek-V4-Pro Preview). Shaded boxes are deep-thinking memos; labels N' mark DT calls that are not counted as environment steps.

the gains. The improvement therefore comes from the joint interface-policy update: the model learns both when to invoke deep thinking and how to use the resulting memo in its next environment action. The same ordering holds on ToolHop and GAIA; on ALFWorld, RL-ReAct is slightly above RL-Gate-only but remains far below either full policy. The lower token cost of RL-ReAct highlights the accuracy-efficiency trade-off rather than a missing control signal. The full 150-step sensitivity analysis in Appendix G further shows that RL-Action performance remains stable across the tested reward initialization and switch thresholds.

5 Conclusion

We compare three deliberative-thinking interfaces—Periodic Thinking, Sel.-Gate, and Sel.-Action—within a ReAct agent loop across six model families and three benchmarks. Selective deep thinking consistently improves over standard

ReAct, but the best interface depends on model capability: Sel.-Action works best for strong proprietary models, whereas Sel.-Gate is more reliable for smaller open-source models. With RL training, this ordering *inverts*: Sel.-Action, initially the weakest inference-time mode on Qwen3-8B, becomes the strongest learned policy and reduces token use. The learned policies also converge on a *plan-early, act-then-adapt* trigger pattern from task-completion reward alone. These results provide practical guidance for designing efficient, controllable deliberation mechanisms in language agents.

Limitations

Our experiments are limited to text-only agent benchmarks. We use one deliberation budget ($k = 3$ for Periodic Thinking), selected on a small development sample, and do not exhaustively tune k or the gate temperature. Our RL experiments are

further limited to one backbone (Qwen3-8B) and one training environment (ALFWorld), because GRPO rollouts over long trajectories are computationally expensive. Thus, it remains open whether the observed interface inversion transfers to larger open-source models or proprietary systems. The RL-Gate setting also has a structural disadvantage: the gate decision and downstream action are generated as separate sub-rollouts rather than as one unified policy. Future work should study unified action spaces for cleaner RL optimization. Finally, we do not run repeated rollouts for formal significance testing, and our deep-thinking prompts are manually written in English. Performance may therefore depend on prompt design and may not directly generalise to non-English agent tasks.

Ethical Considerations

This work does not introduce new data collection or human-subject procedures. We evaluate on publicly released academic benchmarks and use proprietary models through the OpenRouter unified API. The main ethical concern is that deliberation notes may make an agent’s behaviour appear more transparent or reliable than it is: a detailed plan can still be wrong, misleading, or unsupported by the environment. Downstream systems should therefore treat deep-thinking notes as unverified model-generated text, log and monitor their use, and avoid relying on them as explanations in high-stakes settings without additional validation.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback which has greatly improved the paper. The authors acknowledge the financial support from the National Natural Science Foundation of China (Grant Nos. 62673340, 62276171, 62476173, 62532007), the Shenzhen Science and Technology Program (Grant Nos. JCYJ-20240813141503005, JCYJ20240813142610014, ZDCY20250901110940006), and the Major Special Project for Philosophy and Social Sciences Research of the Ministry of Education (Grant No. 2025JZDZ010).

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler.

2024. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI Conference on Artificial Intelligence*.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Do NOT think that much for $2+3=?$ on the overthinking of long reasoning models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*.

DeepSeek-AI. 2026. DeepSeek-V4: Towards highly efficient million-token context intelligence. https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4.pdf.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, et al. 2025. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Aaron Grattafiori et al. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Tao He, Lizi Liao, Yixin Cao, Yuanxing Liu, Ming Liu, Zerui Chen, and Bing Qin. 2024. Planning like human: A dual-process framework for dialogue planning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Xiaoxi Li, Wenxiang Jiao, Jiarui Jin, Guanting Dong, Jiajie Jin, Yinuo Wang, Hao Wang, Yutao Zhu, Jirong Wen, Yuan Lu, and Zhicheng Dou. 2026. DeepAgent: A general reasoning agent with scalable toolsets. In *Proceedings of the ACM Web Conference (WWW)*.

Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. 2025. Reasoning models can be effective without thinking. *arXiv preprint arXiv:2504.09858*.

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

Grégoire Mialon, Clémentine Fourier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations (ICLR)*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on*

- Empirical Methods in Natural Language Processing (EMNLP)*.
- OpenAI, Josh Achiam, Steven Adler, et al. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.
- OpenAI et al. 2024. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*.
- Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*.
- Davide Paglieri, Bartłomiej Cupiał, Jonathan Cook, Ulyana Piterbarg, Jens Tuyls, Edward Grefenstette, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. 2025. Learning when to plan: Efficiently allocating test-time compute for LLM agents. *arXiv preprint arXiv:2509.03581*.
- Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*.
- Qwen Team. 2024. [QwQ: Reflect deeply on the boundaries of the unknown](#). Qwen Blog.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Mrinal Rawat, Ambuje Gupta, Rushil Goomer, Alessandro Di Bari, Neha Gupta, and Roberto Pieraccini. 2025. Pre-Act: Multi-step planning and reasoning improves acting in LLM agents. *arXiv preprint arXiv:2505.09970*.
- Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. 2025. [smolagents: a smol library to build great agentic systems](#). Hugging Face / GitHub.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*.
- Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. 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 (ACL)*.
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*.

- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. ReWOO: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*.
- Shiyu Yan, Xin Yang, Zhen Huang, et al. 2026. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*.
- An Yang et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.
- Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, Tao Gui, Qi Zhang, Xuanjing Huang, and Jiecao Chen. 2025. ToolHop: A query-driven benchmark for evaluating large language models in multi-hop tool use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Yang Yu, Lin Yao, Yuxuan Xie, et al. 2026. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. *arXiv preprint arXiv:2601.01885*.
- Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. 2025a. AdaptThink: Reasoning models can learn when to think. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Shao Zhang, Xihuai Wang, Wenhao Zhang, Chaoran Li, Junru Song, Tingyu Li, Lin Qiu, Xuezhi Cao, Xunliang Cai, Wen Yao, Weinan Zhang, Xinbing Wang, and Ying Wen. 2025b. Leveraging dual process theory in language agent framework for real-time simultaneous human-AI collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. 2024. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.
- King Zhu, Wangchunshu Zhou, et al. 2025. Scaling test-time compute for LLM agents. *arXiv preprint arXiv:2506.12928*.

A Implementation Details

Backbones and decoding. We use six backbones via the OpenRouter unified API or local vLLM checkpoints: GPT-5.2, GPT-4o (OpenAI), DeepSeek-V4-Pro Preview (DeepSeek-AI), Qwen3-8B, Qwen3-32B (Yang et al., 2025) (local vLLM), and Llama-3.3-70B-Instruct (Grattafiori et al., 2024) (OpenRouter model identifier `meta-llama/llama-3.3-70b-instruct`). Proprietary and third-party open-source models (GPT-5.2, GPT-4o, DeepSeek-V4-Pro Preview, Llama-3.3-70B-Instruct) are accessed via the OpenRouter unified API; Qwen3-8B and Qwen3-32B are served locally with vLLM. Open-source backbones are run with their native thinking mode disabled (`reasoning_effort=none`) and temperature 0.0; proprietary backbones use temperature 1.0 with `deep_think_reasoning_effort=low` (the deep-thinking inner call is the only place where proprietary models are permitted any low-effort native thinking, keeping the comparison across modes controlled). The outer agent contract for all modes is line-based `Reason: / Action:` (rather than XML-tagged `<think>`), which we found necessary to keep gate-format parsing robust against orphan `</think>` tokens emitted by some open-source models. The deep-thinking inner format is `Deep think: ... \n Response: ...`; the `Response` portion is the memo persisted to context. All inference experiments are seed-matched across modes for a given backbone (seed 0 throughout).

Benchmark-specific harnesses. All three benchmarks use structurally identical six-mode logic—the same `Thought/Action` outer contract, the same deep-thinking prompt, and the same evaluator-side dispatch (Algorithm 2)—only the environment interface and tool schema differ.

ALFWorld. We use the community TextWorld-based ALFWorld implementation. The agent is evaluated on the *unseen validation* split consisting of all 134 tasks (the full unseen-valid set); no tasks are held out from this split. The environment provides a set of admissible actions at each step, and the agent must select exactly one.

ToolHop. We adopt ToolHop’s native agent framework without modification except for the thinking-mode wrapper and episode loop. Specifically, the tool registry, tool-call parsing, and result formatting follow the benchmark’s reference implementation; only the `Thought/Action` generation step and the evaluator-side DT trigger are replaced by our unified loop.

GAIA. GAIA does not ship a runnable agent harness. We use the community `smolagents` framework (Roucher et al., 2025), which is widely adopted for GAIA evaluation, and evaluate on the 103 text-only items of the public validation set. The thinking-mode wrapper is inserted at the agent’s generation step; all other tool-use and web-search logic follows the `smolagents` default.

Step budgets. ALFWorld: maximum 50 steps per episode. ToolHop: maximum 15 steps. GAIA: maximum 25 steps. Episodes that exceed the budget are counted as failures. RL training rollouts use a shorter budget of 30 steps to reduce wall-clock time; evaluation always uses the full budget.

Sel.-Action schema. The `deep_think` tool is added to the environment tool schema as a parallel tool with no environment side effect. If invoked at step N , it is removed from the schema for step $N+1$, forcing the agent to take a real environment action; it re-enters the schema at step $N+2$. There is no hard cap on the total number of `deep_think` calls per episode; this is governed naturally by the temporary schema ablation.

Sel.-Gate schema. Before each `Reason+Action` turn we issue a short sub-call that returns exactly one JSON field, `deep_think: true | false`. When `true`, a deep-thinking step is generated and the resulting memo is injected into the context before the regular `Reason+Action` turn. For open-source gate models we disable any internal chain-of-thought to keep the gate latency bounded.

B RL Training Details and Reward Analysis

Training dynamics. Only 150 training steps suffice to unlock the small backbone’s ability to drive the selective deep-thinking mechanism; training curves are provided in Appendix B.

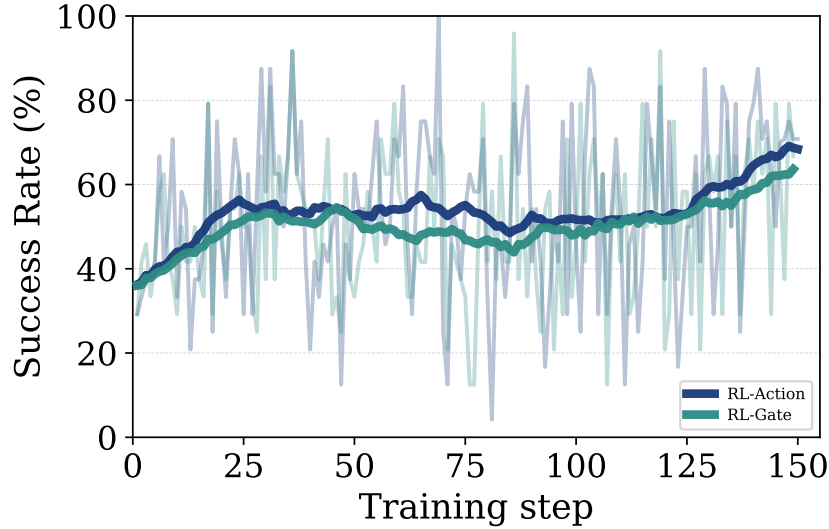


Figure 4: GRPO training curves for Qwen3-8B using only the ALFWorld training split. Both selective interfaces improve steadily and converge within 150 steps.

GRPO hyper-parameters. We train a single LoRA adapter (rank 32, $\alpha = 64$, target all linear layers) on top of Qwen3-8B. Rollout batch size: 8 episodes per update, group size 8 (i.e. 64 rollouts per batch). PPO mini-batch size 32. Learning rate 1×10^{-6} ; KL loss coefficient 0.01 (`low_var_kl`); invalid-action penalty coefficient 0.2. Validation temperature 0.4; training rollout temperature 1.0. Total 150 training steps. Random seed: `env.seed=0`. Success-rate moving-average window: 128 episodes. Adaptive deep-thinking reference μ^* initialised at 5. Efficiency coefficient $\lambda_{\text{tok}} \in \{0, 0.05\}$ with a hard switch at $\text{SR} = 0.6$. Training environment: ALFWorld training split (≈ 300 tasks). RL results reflect a single training run; repeated runs were not performed due to the substantial compute cost of GRPO rollouts over long agent trajectories. The resulting RL-Action policy is evaluated zero-shot on the ALFWorld unseen-134 split as well as on the full ToolHop and GAIA evaluation sets, with no further training. Code is released at <https://github.com/cherish30/selective-deliberation-training>. The full 150-step sensitivity analysis for the two key reward hyperparameters is reported in Table 2.

Attribution ablations (RL-ReAct and RL-Gate-only). To separate the gains of RL-Action from generic policy optimization, we train two controls under the identical reward (Table 1, last two RL rows). (i) *RL-ReAct* fine-tunes the plain ReAct policy with no separate `deep_think` channel: under the token-efficiency penalty it degenerates, suppressing its Thought to evade the penalty (ALFWorld output collapses to 1468 tokens) and reaching only 55.2% ALFWorld SR; the loss of reasoning surfaces on transfer, where ToolHop drops to 29.7% (tokens 491) and GAIA to 9.7%, far below RL-Action (67.9/48.2/21.4). Thus the gains are not explained by generic action-policy RL. (ii) *RL-Gate-only* learns the binary `deep_think` decision while freezing the downstream Reason+Action model. It improves ALFWorld SR only modestly ($52.2 \rightarrow 54.5$, versus 61.9 for joint RL-Gate) and does not reduce total tokens: training the gate alone, with the downstream policy frozen, brings little task-level improvement, indicating that controller learning and the downstream policy must be co-adapted to take effect. This motivates RL-Action—which co-adapts the deliberation decision and the environment action in a single decoupled pass—as the primary learned policy in the main text.

Computing infrastructure and model sizes. All RL training and open-source inference experiments are run on $2 \times$ NVIDIA A800 (80 GB) GPUs. RL-Action training (Qwen3-8B, ALFWorld, 150 training steps) took 78.2 GPU-hours; RL-Gate training under identical settings took 84.6 GPU-hours. Proprietary backbones (GPT-5.2, GPT-4o, DeepSeek-V4-Pro Preview) are accessed via the OpenRouter unified API; their parameter counts are not publicly disclosed. Open-source backbone parameter counts are encoded in their model names: Qwen3-8B (8 B), Qwen3-32B (32 B), and Llama-3.3-70B-Instruct (70 B). The RL-trained adapter adds LoRA rank-32 parameters on top of Qwen3-8B (≈ 85 M additional parameters).

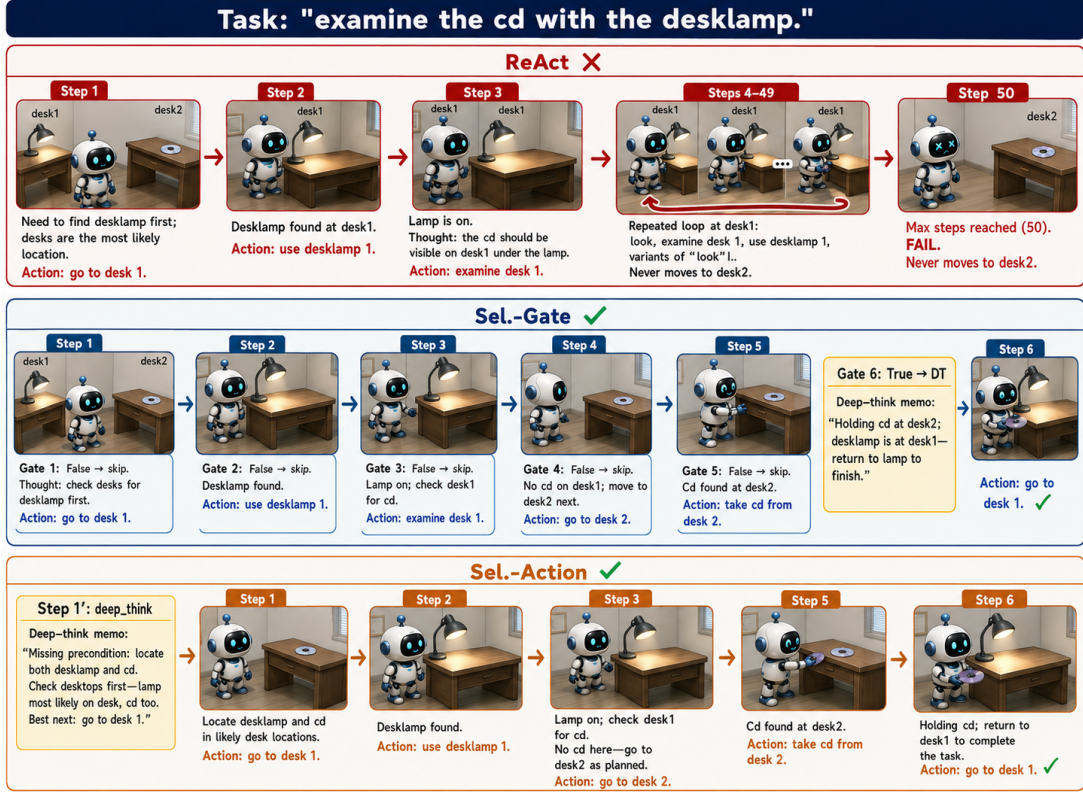


Figure 5: Visual walkthrough of the case study in Figure 3: ReAct loops at desk 1 for 47 steps; Sel.-Gate fires deep thinking reactively at step 6; Sel.-Action fires proactively at step 1' before any search begins. Both selective variants succeed in 6 steps.

Software versions. All experiments use PyTorch 2.6.0, vLLM 0.8.5, Transformers 4.51.1, and PEFT 0.19.1 for local inference and RL training. ALFWorld 0.4.2 provides the embodied environment; smolagents 1.25.0 provides the GAIA agent harness. ToolHop is evaluated against its official reference implementation with no third-party dependencies beyond the standard Python stack.

Reward magnitude analysis. A quantitative check that no single penalty dominates the unit task signal. A representative deep-thinking call contributes roughly 500 tokens, or ≈ 0.0073 to the efficiency penalty—negligible in mixed-outcome groups (within-group success-rate std ≈ 0.44) but a clean $\sim 2\sigma$ ordering signal in high-success-rate groups. The total efficiency penalty saturates around 0.15 at 12k tokens, comfortably below the 1.0 task reward. The DT-quality penalty caps at a few percent of the task reward except in extreme over-thinking (> 5 calls past μ^*), where it begins to dominate. This staged signal stack lets GRPO discover deep thinking from the outcome alone in early training and refine its allocation once the task is mostly solved.

C Case Study Traces

We present two representative case studies to illuminate how selective deep thinking affects agent behaviour at the trajectory level: one from inference-time deliberation (Figure 3) and one from RL-trained deliberation (Figure 6). Figure 5 and Figure 7 provide spatial walkthroughs of the same two episodes.

C.1 Case Study 1: Inference-Time Selective Deliberation

Task. The agent must “*examine the cd with the desk lamp*” in an ALFWorld household environment (trial idx = 20, backbone: DeepSeek-V4-Pro Preview). The task requires two simultaneous sub-goals: (i) locate and activate the desk lamp, and (ii) locate the cd and bring it under the lit lamp. The lamp is on desk 1; the cd is on desk 2, which requires a separate navigation step not prompted by desk 1’s contents.

ReAct (FAIL, 50 steps). ReAct reaches the desk lamp in two steps and turns it on, but immediately anchors on desk 1 as the cd’s location. From step 4 onward it enters a 47-step loop alternating between look, examine desk 1, and use desk lamp 1, each time re-generating a locally valid thought such as “the lamp is on; the cd should be visible on desk 1”. The thought at each step is *contextually coherent* given the immediately preceding observation, which is why standard chain-of-thought cannot detect the loop: no single step looks wrong in isolation. What is missing is a *meta-cognitive* vantage point that can ask “why has the same action failed 10 times in a row?” and formulate a different search strategy. ReAct exhausts its 50-step budget without ever visiting desk 2.

Sel.-Gate (SUCCESS, 6 steps). The gate remains FALSE through steps 1–5, allowing the agent to make efficient forward progress: go to desk 1, activate the lamp, examine desk 1 (no cd found), move to desk 2, and take cd 3. The gate fires TRUE exactly once, at step 6, when the observation “you pick up the cd 3” signals a new, potentially non-obvious sub-goal transition. The full deep-thinking output at this point is:

thinking: “We have the cd in inventory but the lamp is on desk 1. Task wants examining cd with desk lamp, so we likely need to be at desk 1/lamp context then examine cd. Next move is to return to desk 1; if needed, ensure lamp is on and then examine cd.”

memo: “Holding cd at desk 2; desk lamp is at desk 1—return to lamp to finish.”

The `thinking` field performs the spatial reasoning (lamp is at desk 1, agent is at desk 2); the `memo` distills this into an actionable precondition and a concrete next step. The agent navigates back to desk 1 and completes the task. This illustrates the gate’s *reactive* trigger model: it monitors for sub-goal transitions and fires *after* a change in agent state whose implications for the remaining plan are non-trivial.

Sel.-Action (SUCCESS, 6 steps). Before issuing any action, the agent invokes `deep_think` at step 1’. The full output is:

thinking: “Need both cd and desk lamp; likely on desks or inside drawers/shelves. Start with desks since lamp often sits there and cd could be on top. If not on desks, sweep drawers then shelves, then safe/hamper/garbage can. After acquiring both, use ‘examine cd with desk lamp’ action sequence. Proceed systematically.”

memo: “Missing precondition: locate both desk lamp and cd. Check desktops first—lamp most likely on desk, cd too. Best next: go to desk 1.”

The `thinking` field sketches a complete fallback search order (desks → drawers → shelves → safe); the `memo` compresses this into a single actionable directive that the policy executes directly. This single deliberation act decomposes the task into two independent sub-goals before any action is taken, effectively preventing the anchoring failure that dooms ReAct. Steps 1–6 then proceed without any further DT call: go to desk 1, activate lamp, examine desk 1 (no cd), go to desk 2, take cd, return to lamp. The proactive memo eliminates the need for reactive correction later.

Key contrast. The critical difference is *when* deliberation fires: Sel.-Action reasons before any action and therefore builds a complete plan; Sel.-Gate discovers the missing sub-goal reactively at the last transition. Both succeed in the same number of steps (6), but they represent qualitatively different uses of deep thinking—planning-ahead versus state-change monitoring. ReAct, lacking either mechanism, cannot recover from its initial anchoring error.

C.2 Case Study 2: RL-Trained Deliberation Policy

Task. The agent must “find two toilet paper and put them in cabinet” in an ALFWorld household (trial idx = 91, backbone: Qwen3-8B). This is a multi-object retrieval task: the agent must locate both toilet papers, store them in the target cabinet, and return for the second item without losing track of what has already been placed. Both toilet papers are on `countertop 1`; the cabinet is elsewhere in the room.

Task: "find two toiletpaper and put them in cabinet."	
Sel.-Action (base, Qwen3-8B) ✗	Sel.-Action (RL-action, Qwen3-8B) ✓
Obs 1: Room: toilet1, countertop1, cabinets, sinkbasin... Th 1: Find 2 toiletpapers; hanger is the most likely location. Act 1: <code>go to toiletpaperhanger 1</code> Obs 2: toiletpaperhanger1: empty. Th 2: Nothing here; check toilet next. Act 2: <code>go to toilet 1</code> Obs 3: toilet1: soapbar3, toiletpaper3 Th 3: Found tp3--take it, then find the second one. Act 3: <code>take toiletpaper 3 from toilet 1</code> Obs 4: You pick up the toiletpaper3. Th 4: Have tp3; the second one must be in a cabinet. Act 4: <code>go to cabinet 1</code> ... Steps 5-13: searches cabinets1-3, finds nothing ... Obs 14: countertop1: candle, soapbottle, toiletpaper1... Th 14: That's the second toiletpaper I need! Act 14: <code>go to cabinet 1</code> Obs 15: cabinet1: candle, cloth (no toiletpaper). Th 15': Still nothing; reflect before next move. Act 15': <code>deep_think</code> Take toiletpaper1 from countertop1--it is visible and in reach. Th 15: Cabinet1 is empty; try cabinet2. Act 15: <code>go to cabinet 2</code> ... Steps 16-17: DT fires 2X more--similarly ignored ... Obs 18: countertop1: candle, soapbottle, toiletpaper1... Th 18: Let me examine the countertop more carefully. Act 18: <code>examine countertop 1</code> Obs 19: countertop1: candle, soapbottle, toiletpaper1... Th 19: Let me examine the countertop more carefully. Act 19: <code>examine countertop 1</code> Steps 18-24 (X7, identical): sees toiletpaper1 on countertop1; DT fires 6X urging "take tp1" but agent keeps examining. Never picks it up. ✗ MAX STEPS (50). FAIL.	Obs 1: Room: toilet1, countertop1, cabinets, sinkbasin... Th 1: Toiletpaper is usually kept near the toilet; go there first. Act 1: <code>go to toilet 1</code> Obs 2: toilet1: soapbar3, toiletpaper3 Th 2': Found one; plan before acting. Act 2': <code>deep_think</code> Take toiletpaper3 from toilet1 and store it in a cabinet first; then search for the second with empty hands. Th 2: Store tp3 in cabinet4 first. Act 2: <code>take toiletpaper 3 from toilet 1</code> Obs 3: You pick up the toiletpaper3. Th 3: Bring tp3 to cabinet4. Act 3: <code>go to cabinet 4</code> Obs 4: cabinet4: closed. Th 4: Open cabinet4. Act 4: <code>open cabinet 4</code> Obs 5: cabinet4: cloth2... Th 5: Place tp3 here; then search for the second. Act 5: <code>move toiletpaper 3 to cabinet 4</code> Obs 6: Moved toiletpaper3 to cabinet4. Th 6: First item stored; return to toilet to check for the second. Act 6: <code>go to toilet 1</code> Obs 7: toilet1: soapbar3 only (no more toiletpaper). Th 7': Toilet is empty--second must be elsewhere. Act 7': <code>deep_think</code> The second toiletpaper is likely on the countertop. Go to countertop1 next. Th 7: Heading to countertop1. Act 7: <code>go to countertop 1</code> Obs 8: countertop1: candle3, toiletpaper1... Th 8: There it is--take toiletpaper1. Act 8: <code>take toiletpaper 1 from countertop 1</code> Obs 9: You pick up the toiletpaper1. Th 9: Both collected; bring tp1 to cabinet4. Act 9: <code>go to cabinet 4</code> Obs 10: cabinet4: cloth2, toiletpaper3... Th 10: Place tp1 here--task done. Act 10: <code>move toiletpaper 1 to cabinet 4</code> ✓

Figure 6: Case study (ALFWorld, trial idx = 91, Qwen3-8B). Shaded boxes are deep-thinking memos; labels N' mark DT calls (act: `deep_think`) that are not counted as environment steps.

Base Qwen3-8B policy (FAIL). The base policy places the first toiletpaper (toiletpaper3) correctly but then fails to recognise that countertop1 still holds the second one. Instead, it embarks on an exhaustive cabinet-by-cabinet search, visiting locations it has already checked, while the second object remains in plain sight. Deep thinking fires *six times* during this search; each time the deliberation correctly diagnoses the problem, yet the immediately following action ignores it. A representative example is DT #2 (env step 16):

thinking: “The agent has identified a second toiletpaper (toiletpaper1) on the countertop but has not yet picked it up. The next logical step is to take this toiletpaper before attempting to move it to the cabinet.”

memo: “Take toiletpaper1 from countertop1—it is visible and in reach.”

The thinking field reasons correctly; the memo issues a clear imperative. Yet the action taken is look—a complete non-sequitur. The same pattern repeats across all six DT calls: accurate deliberation, zero compliance. This dissociation reveals that without RL, the `deep_think` outputs are generated but exert no real influence on the policy’s action selection.

RL-trained policy (SUCCESS, 10 steps). The RL-trained policy invokes deep thinking only twice and acts on both outputs immediately. At step 2, upon spotting `toiletpaper 3` on `toilet 1`, it triggers DT proactively:

thinking: *“I found toiletpaper 3 on toilet 1. The task requires two toiletpapers in a cabinet. I should pick this one up now, then immediately store it in a nearby cabinet to free my hands before searching for the second. Cabinets are the destination anyway, so storing the first one early avoids a double trip later.”*

memo: *“Take toiletpaper 3 from toilet 1 and store it in a cabinet first; then search for the second with empty hands.”*

The agent executes this plan exactly: take, store in `cabinet 4`, return. At step 7, finding `toilet 1` now empty, it triggers DT again to re-plan:

thinking: *“Toilet 1 is now empty—the second toiletpaper is not here. I have already stored the first one in cabinet 4. The most likely remaining location for a toiletpaper is the countertop, which I have not checked yet.”*

memo: *“The second toiletpaper is likely on the countertop. Go to countertop 1 next.”*

The agent navigates directly to `countertop 1`, picks up `toiletpaper 1`, and completes the task in 10 steps.

What RL changes. The `thinking` and `memo` quality is comparable across both policies—the base model also generates correct diagnoses. What differs is *memo compliance*: the RL-trained policy acts on every memo it generates, while the base policy ignores all six. RL training instils two complementary changes: (1) a *store-first* procedural prior that triggers deliberation at the right moment rather than after repeated failures, and (2) genuine compliance so that deliberation outputs directly guide the next action. Together these changes make deliberation both *selective* and *effective*: fewer calls, each with higher causal impact.

D Supplementary Analysis

Token–accuracy frontier. Plotting average tokens against success rate across the six modes reveals two Pareto fronts. Proprietary models cluster on a front anchored by ReAct and advanced by Sel.-Action. Open-source models trace a front where Sel.-Gate sits closest to the Pareto-optimal corner; Periodic Thinking offers the highest success rate at higher cost and Always Thinking is dominated almost everywhere.

What does deep thinking write? Inspecting deep-thinking traces we observe three recurring patterns: (i) sub-goal book-keeping (“I still need to find the saltshaker”), (ii) dead-end detection (“the cabinet I checked twice is empty; try the drawers”), and (iii) multi-hop tool planning (“call `get_company_by_ceo`, then chain with `founded_year`”). Reflexion-style post-hoc reflections only appear after a failure; deep thinking intervenes *during* the trajectory and prevents some of the failure modes Reflexion would later have to explain.

Memo quality varies considerably across episodes. The three excerpts below are drawn verbatim from GPT-5.2 Sel.-Action runs on GAIA Level-2; each card shows the task context, the criterion that determines quality, and the resulting memo.

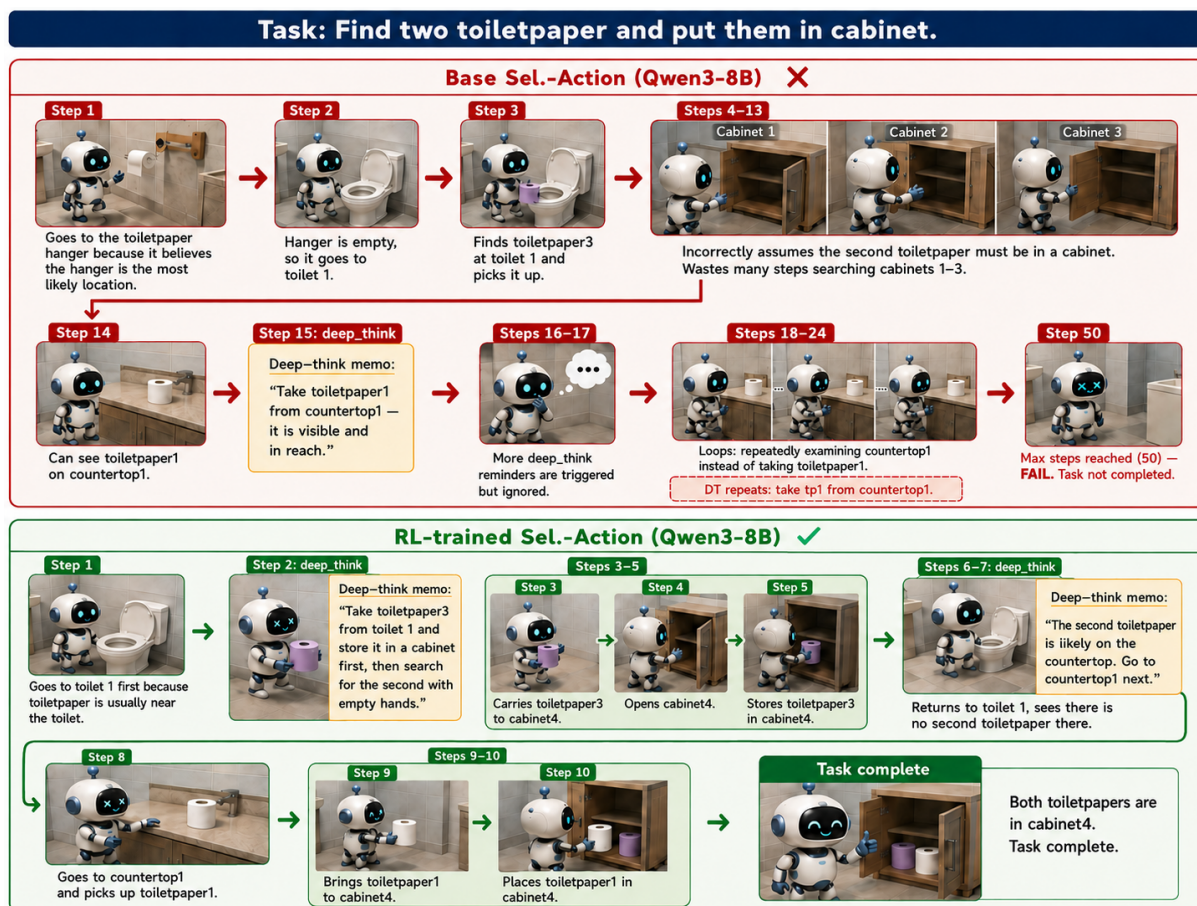


Figure 7: Visual walkthrough of Figure 6: the base Qwen3-8B policy ignores six DT memos and loops at countertop 1; the RL-trained policy invokes deep thinking only twice—at step 2 to plan a store-first strategy, and at step 7 to redirect to the countertop—completing the task in 10 steps.

HIGH QUALITY

✓ episode solved

Task: Identify who served as Prime Minister in April 1977, but only after verifying whether the NIV Bible renders Esther 1:1 as “India to Cush” or begins with “Susa”. The answer depends on which phrasing is correct, so the model must resolve a translation fact before the historical lookup.

Memo: “The task hinges on whether the NIV renders Esther 1:1 as India to Cush or opens with Susa; the ordering must be verified from an actual NIV source before looking up the April-1977 Prime Minister.”

Why high quality: The memo names the exact prerequisite check (*which NIV rendering?*) and prescribes a concrete, verifiable next action before the main lookup. Without this anchor the agent would guess the translation and likely retrieve the wrong PM.

MEDIUM QUALITY

× episode failed

Task: Report per-trip passenger counts for a specific SFRTA (South Florida Regional Transportation Authority) commuter route. Public ridership dashboards aggregate by day or month; the model must find or infer whether trip-level data is published at all.

Memo: “Trip-level passenger counts typically live in internal APC exports, not public ridership summaries; the key missing piece is whether SFRTA publishes per-train data openly.”

Why medium quality: The memo correctly diagnoses the data-access barrier (APC exports vs. public summaries) and identifies the key open question, but stops there. It provides no candidate tool calls, no fallback sources, and no next action—so the agent stalls at the same dead end.

LOW QUALITY

× episode failed

Task: A multi-step GAIA Level-2 research question requiring several tool calls to locate and cross-validate a specific historical statistic.

Memo: “Use a simple framework: (1) restate the exact deliverable and constraints, (2) list the data and tools needed, (3) write an ordered plan: clarify → gather → execute → verify.”

Why low quality: The memo ignores all task-specific content and outputs a domain-agnostic planning template that could apply to any question. It carries no information the agent did not already have from the system prompt, so invoking deep thinking was wasted compute. This pattern mirrors the overactive-DT failure in category (b) of Appendix F.

Cost of the gate sub-step. The gate adds a fixed overhead of ~ 20 tokens per turn (the `deep_think: false` case). On open-source ALFWorld this overhead is more than offset by avoided unnecessary deliberations, yielding the -27% token figure in the main text. On strong models the overhead is comparable to the saved deliberations, which is why Sel.-Action—with no gating overhead—wins in that regime.

Why Sel.-Action fails at inference on small models. Tracing Qwen3-8B rollouts reveals two failure modes: (a) *under-thinking*—the model rarely emits `deep_think`, plausibly because deliberation-as-action is rare in pre-training—and (b) *misplaced thinking*—it issues `deep_think` in trivial states (e.g. an empty hallway), wasting tokens. Sel.-Gate mitigates both by reducing “think or not” to a tiny structured classification; RL-Action mitigates both by directly optimising the placement policy, which is what produces the interface inversion of §4.2.

E Prompts and Mode Pseudocode

All six modes share a common outer contract (`Thought : /Action :`) and the same inner deep-thinking call. We first give control-flow pseudocode, then list the shared prompts once and show only the mode-specific delta for each mode.

E.1 Six-Mode Step-by-Step Flow

Each block traces one full episode step for that mode, using $\rightarrow$ to show how control flows between the evaluator, the inner deep-think model, and the outer agent. `InnerDT(obs, H, memo)` calls the shared inner system prompt (Section E.3) and returns a memo string.

Mode 3 -- Periodic Thinking (k = 3)

```
-----
Each step t:
  if t > 0 and t % 3 == 0
    --> InnerDT(obs, H, memo) --> memo (injected into context)
  obs --> outer_model(obs, history + memo, sys_prompt)
    --> "Thought: ... / Action: <env_action>"
    --> env.step(env_action) --> obs', done
```

Mode 4 -- Always Thinking

```
-----
Each step t:
  (unconditional) --> InnerDT(obs, H, memo) --> memo (injected into
context)
  obs --> outer_model(obs, history + memo, sys_prompt)
    --> "Thought: ... / Action: <env_action>"
    --> env.step(env_action) --> obs', done
```

Mode 5 -- Sel.-Action (deep_think as a parallel tool)

```
-----
Step N [schema = env_tools + {deep_think}]:
  obs --> outer_model(schema_N, history + memo, obs)

  branch A -- model emits deep_think:
    --> "Thought: ... / Action: deep_think"
    --> InnerDT(obs, H, memo) --> memo (injected into context)
    |
    | schema_N+1 = env_tools (deep_think removed for 1 step)
    |
    --> outer_model(schema_N+1, history + memo, obs) [re-query, same obs]
    --> "Thought: ... / Action: <env_action>"
    --> env.step(env_action) --> obs', done

  branch B -- model emits env_action directly:
    --> "Thought: ... / Action: <env_action>"
    --> env.step(env_action) --> obs', done
    (schema unchanged; deep_think still available at step N+1)

Step N+1 [schema = env_tools, deep_think suppressed after branch A]:
  obs --> outer_model(schema_N+1, history + memo, obs)
    --> "Thought: ... / Action: <env_action>"
    --> env.step(env_action) --> obs', done

Step N+2 [schema = env_tools + {deep_think} restored]:
  ... repeat from Step N
```

Mode 6 -- Sel.-Gate (Sel.-Gate, two-phase per step)

```
-----
Each step t:
  [Phase 1 -- gate decision, /no_think, budget-capped at 24 tokens]
  gate_facts = {memo_age, last_action, invalid_flag, repeat_count}
  obs --> gate_model(obs, H, memo, gate_facts, /no_think)
    --> "Deep_think: true" or "Deep_think: false"

  if gate == true:
    --> InnerDT(obs, H, memo) --> memo (injected into context)

  [Phase 2 -- action emission, full generation]
  obs --> outer_model(obs, H + memo, sys_prompt_phase2)
    --> "Thought: ... / Action: <env_action>"
    --> env.step(env_action) --> obs', done
```

E.2 Mode Dispatch and Episode Loop Pseudocode

Algorithm 1 — Evaluator-Side Trigger (ShouldForceDT)

Determines whether the evaluator should inject a forced deep-think call before the outer agent's turn. Used by Periodic (k -step schedule) and Always (unconditional). Returns a (force, reason) pair.

```
Input:  mode m, history H (list of prior steps), step index t,
        period k (Periodic only)
Output: (force: bool, reason: str)

# --- Always: fire every step ---
if m == Always:
    return (True, "always_every_step")

# --- Periodic: fire at multiples of k (k=3 in experiments) ---
if m == Periodic:
    if t > 0 and t % k == 0:
        return (True, "periodic_every_{k}_steps")
    return (False, "")

# --- Sel.-Action / Sel.-Gate: no evaluator-forced DT ---
return (False, "")
```

Algorithm 2 — Episode Loop

Shows how the evaluator-side trigger, inner DT call, and mode-specific outer call compose for all six modes within one episode.

```
Input:  mode m, environment env, max steps T,
        period k (Periodic only)

H      = []          # trajectory history
memo   = None        # latest deep-think memo (persisted across steps)
obs    = env.reset()

for t in range(T):

    # -- Evaluator-side forced DT (Periodic / Always) -----
    force, reason = ShouldForceDT(m, H, t, k)
    if force:
        memo = InnerDT(obs, H, memo, trigger=reason)
        # InnerDT calls the inner deep-think model and returns the memo text

    # -- Mode-specific outer call -----
    if m == Sel-Gate:
        # Phase 1: gate decision (max 24 tokens, no_think)
        gate = GateCall(obs, H, memo)          # returns True or False
        if gate == True:
            memo = InnerDT(obs, H, memo, trigger="two_phase_gate_true")
        # Phase 2: Thought + Action (full generation with updated memo)
        action = ActionCall(obs, H, memo, dt_just_called=gate)

    elif m == Sel-Action:
        # Outer agent can choose Action: deep_think voluntarily
        action = OuterCall(obs, H, memo)        # may return "deep_think"
        if action == "deep_think":
            memo = InnerDT(obs, H, memo, trigger="model_requested")
            action = OuterCall(obs, H, memo)     # re-query; deep_think removed

    else:
        # Periodic / Always: single outer call
        action = OuterCall(obs, H, memo)

    # -- Environment step -----
    obs, done = env.step(action)
    H.append({"action": action, "observation": obs, "memo": memo})
    if done:
        break
```

E.3 Shared Prompts

Base Outer System Prompt (all modes)

All six modes start from the same outer system prompt prefix, which is then extended with a mode-specific format specification.

```
You are an expert ALFWorld agent.
Follow the required response format exactly.
Do not output extra text outside the required response format.
```

Inner Deep-Think System Prompt (Periodic / Always / Sel.-Action / Sel.-Gate)

When a deep-thinking step is triggered, the inner model receives the current observation and trajectory context under this system prompt. Its output has two sections: `Deep think:` (private reasoning, discarded after the call) and `Response:` (a concise planning memo written back into the outer agent's context).

```
# Role
You are an internal deep-thinking tool for an ALFWorld agent.
You receive the same ALFWorld task context, recent trajectory context,
current observation, and current admissible actions.

# OutputFormat
Reply with exactly two labeled sections in this order:
  Deep think: [your internal reasoning and deliberation]
  Response:   [your concise action memo]

# DeepThink
Use "Deep think:" for internal reasoning: analyze the current state, what has been
  private
  and not shown to the outer agent. Do not enumerate a long search tree; reason
  toward
  a conclusion rather than listing possibilities.

# Response
Use "Response:" for a compact memo written back into later context as the outer
agent's guide. This section is mandatory and must always be present. Write 2-4
direct natural-language sentences:
  - Sentence 1: the key conclusion, blocker, or missing precondition.
  - Sentence 2: the immediate high-level direction for the next few steps.
  - Optional Sentence 3: what progress signal should confirm this direction,
    or the best current admissible next action if one is clear.
  - Optional Sentence 4: when the outer agent should stop following this direction
    and reconsider.
Do not start "Response:" with filler such as "Okay", "Let's", or a restatement of
the task. If one admissible action clearly best advances the plan, name its exact
text; otherwise describe the proxy move at a higher level. Do not use bullet
  points,
  numbered lists, or a literal action script.
```

E.4 Periodic and Always: Shared Per-Step Guidance Template

Periodic Thinking and Always Thinking share the same outer system-prompt suffix and per-step guidance template; only the mode-specific `# Rules` lines differ.

System prompt suffix (all four modes)

```
# OutputFormat
Reply in exactly two lines.
```

Line 1 must start with 'Thought:' and contain a brief task-relevant local judgment.
Line 2 must start with 'Action:' and contain exactly one admissible action.

Per-step guidance template (all four modes)

```
# Task
Now it's your turn to take an action.

# OutputFormat
Reply in exactly two lines:
  Thought: brief task-relevant reasoning about the current observation and next
           move.
  Action:  exactly one admissible environment action for the current step.

# Rules
- Keep Thought short and local.
- Track which locations you have already searched to avoid revisiting them.
- If a task-level atomic action is already admissible and advances the goal,
  prefer it over extra setup actions.
- Only actions in the current admissible list are executable; if your intended
  move is not listed, choose one that can reveal or enable it.
[Mode-specific additions below]
```

Mode-specific # Rules additions:

- **Periodic ($k=3$):** “A separate deep-thinking tool is triggered every k steps by the evaluator. Use the latest deep-think note as your primary guide when present. Do not output `deep_think` yourself.”
- **Always:** “A deep-thinking tool has already run this step; use its note as the main planning source. Do not output `deep_think` yourself.”

E.5 Mode 5 — Sel.-Action (Sel.-Action)

Deep thinking is a *callable action*: the model may emit `Action: deep_think` at any step, whereupon the evaluator runs the inner DT call, injects the resulting memo, and re-issues the same step to elicit a real environment action. `deep_think` is removed from the action space for one step after use (step $N+1$) to prevent back-to-back calls.

System prompt suffix (when `deep_think` is available)

```
# OutputFormat
Reply in exactly two lines.
  Line 1 must start with 'Thought:' and contain a brief task-relevant local
    judgment.
  Line 2 must start with 'Action:' and contain exactly one admissible action from
    the
    prompt, or the exact token deep_think.
```

Per-step guidance (when `deep_think` is available)

```
# Task
Now it's your turn to take an action.

# OutputFormat
Reply in exactly two lines:
  Thought: a brief current-state judgment about the observation and next move.
  Action:  one admissible environment action for the current step, or deep_think.

# Rules
- Keep Thought brief; usually no more than 3 short sentences.
- Use deep_think as the main place for task-level thinking and planning.
- A deep-think note is not a literal action script; translate it into the best
```

```

    current admissible action rather than restating it in Thought.

# WhenToUseDeepThink
Choose Action: deep_think when:
- the next grounded move is unclear,
- recent steps have repeated or failed without progress,
- the current observation contradicts the latest note,
- a short local judgment is not enough to choose one admissible action.
Choose a direct environment action when one concrete admissible action clearly
advances the current task without further planning.

# Examples
Thought: The last two steps both failed; I need a better plan.
Action:  deep_think

Thought: The note says check closed containers; cabinet 2 is available.
Action:  open cabinet 2

```

Per-step guidance (when `deep_think` is *not* available, i.e. step $N+1$ after use): same shared template as Modes 1–4 (§E.4), with one additional rule appended: “*deep_think is not available on this step; use the latest deep-think note above as your guide and execute one concrete action. Do not repeat look or inventory unless the note or observation specifically calls for that check.*”

E.6 Mode 6 — Sel.-Gate (Sel.-Gate, two-phase)

Each step is split into two strictly sequenced sub-calls: **Phase 1** emits only a one-line gate decision (`Deep_think: true/false`), budget-capped at 24 tokens, with native chain-of-thought disabled (`/no_think`). If the gate fires, the inner DT call is executed and the memo is injected before **Phase 2**, which produces the normal Thought+Action output.

Phase 1 — Gate Decision

System prompt: same base outer system prompt (§E.3), with the gate-specific `# OutputFormat` replacing the standard one:

```

# OutputFormat
/no_think
Reply with exactly one line and nothing else.
The line must be exactly 'Deep_think: true' or 'Deep_think: false'.
Default to 'Deep_think: false'. Only choose 'true' when a specific gate signal
applies. Do not output Thought, Action, <think>, <deep_think>, or any other
content.

```

Per-step gate instruction (contextualised with live gate facts — last action, memo age, consecutive repeat count)

```

# Context
Phase 1 of 2 -- gate decision only.

# GateFacts
- Latest memo:          from step {memo_step} (current step {t}, age {memo_age})
- Last action:          "{last_action}"
- Last action invalid: {true|false}
- Last action repeated consecutively: {n} time(s)

# OutputFormat
Reply with EXACTLY one line: 'Deep_think: true' or 'Deep_think: false'.
Default: 'Deep_think: false'.
Calling deep_think is expensive; use it sparingly (target ~1 in every 4-6 steps).
[If memo is fresh (age <= 2): A deep-think note from {n} step(s) ago is in
context;
strongly prefer 'Deep_think: false' unless its plan is clearly broken.]

```

```
# WhenTrue
Choose 'Deep_think: true' ONLY when a concrete signal applies:
- the same action has repeated 2+ times consecutively with no progress,
- the previous action was invalid AND the current observation gives no obvious
  recovery,
- the latest deep-think note directly contradicts the current observation,
- the task phase just changed (target found / state change succeeded / new
  sub-goal needed) and no note covers the new phase.

# WhenFalse
Otherwise choose 'Deep_think: false'. Routine search, taking the next admissible
action implied by the latest note, or simple state changes (open / take / put /
examine) where one action obviously advances the task -- all are 'false'.
Do not output Thought, Action, <think>, <deep_think>, explanations, or any other
content.
```

Phase 2 — Action Emission

If the gate returned `true`, the inner DT call has already run and the fresh memo is in context before Phase 2 is issued.

System prompt: same base outer system prompt as all modes (§E.3), with the following # `OutputFormat` and # `Rules` suffix added:

```
# OutputFormat
Reply in exactly two lines.
  Line 1 must start with 'Thought:' and contain a brief task-relevant local
  judgment.
  Line 2 must start with 'Action:' and contain exactly one admissible action from
  the
  prompt.

# Rules
- Keep Thought brief; usually no more than 3 short sentences.
- Do not output deep_think; the gate already decided this step.
- Do not output a visible <think> block.
```

Per-step action instruction (shared; gate result noted in # `Context`)

```
# Context
Phase 2 of 2 -- action emission.
[gate=true: A fresh deep-think note was just produced above.]
[gate=false: The gate decided no deep-think this step.]

# OutputFormat
Reply in exactly two lines:
  Thought: a brief judgment on the current observation and next move.
  Action: one admissible environment action for the current step.

# Rules
- Keep Thought brief; usually no more than 3 short sentences.
- If a fresh note is present (gate=true), treat it as the primary guide; pick the
  best admissible action that advances its plan rather than restating it
  literally.
- If no new note was produced (gate=false), use any prior memo as high-level
  guidance.
- Do not output deep_think; the gate already decided this step.
- Do not output a visible <think> block.
```

F Failure Case Analysis

We analyse 43 failed episodes from the Qwen3-8B RL-Action evaluation on ALFWorld². All failures exhaust the 50-step budget; none terminate early. We identify three modes.

(a) DT-front-loaded, then navigation loop ($n = 35$, 81.4%). `deep_think` fires at step 1–2, producing a valid plan, then the agent oscillates between two locations (e.g., `desk 1` $\leftrightarrow$ `shelf 3`) without re-invoking `deep_think`. The following trace, extracted verbatim from an RL-Action evaluation episode, illustrates the pattern:

```
Task: look at mug under the desk lamp.

t=1  Thought: The task requires examining a mug under the desk lamp.
      No desk lamp is visible from the starting position;
      desk 1 is the most likely candidate.
      Action: go to desk 1
      Obs:    You arrive at desk 1. On the desk you see a desk lamp 1,
              a creditcard 1, a laptop 1, and two pencils.

t=2  Thought: I can see desk lamp 1 on desk 1. To examine the mug
      under it I first need to turn the lamp on.
      Invoking deep_think to plan carefully.
      Action: deep_think [model requests deliberation]
      Memo:  "use desk lamp 1. Continue with the most grounded next
              direction from the current observation. If the observation
              changes, the last action fails, or similar actions start
              repeating without progress, call deep_think again."

t=3  Thought: The memo says to use desk lamp 1. Following the plan.
      Action: use desk lamp 1 [lamp turns on]

t=4  Thought: Lamp is on. The mug is not on desk 1; checking shelf 3.
      Action: go to shelf 3 [drifts; memo not re-consulted]
t=5  Thought: Mug not found at shelf 3. Returning to desk 1.
      Action: go to desk 1
t=6  Thought: Still no mug on desk 1. Trying shelf 3 again.
      Action: go to shelf 3
t=7  Action: go to desk 1
      ... (desk 1 <-> shelf 3 oscillation; deep_think never re-invoked
           despite the memo's explicit instruction to do so) ...
t=50 [budget exhausted; episode fails]
```

The memo explicitly instructs the agent to re-invoke `deep_think` upon repeated failure, yet the policy never does so: under episode-level binary reward, the gradient signal for a loop-breaking DT call and for the silent suppression of one are identical (both failed episodes yield reward 0). The memo correctly targets use desk lamp 1, but the agent cannot locate the mug and bounces between `desk 1` and `shelf 3` for the final 47 steps.

(b) Overactive DT with degraded output ($n = 4$, 9.3%). `deep_think` fires on nearly every step (≤ 45 of 50) but produces only shallow one-line directives; the context window fills with redundant entries and the agent degenerates to repeating `look`. *Example*: “put a clean mug in coffeemachine.” DT fires at steps 2, 6–50; all memos output a single navigation micro-step; the final 10 actions are `look`.

(c) Wrong-target memo ($n = 4$, 9.3%). The memo encodes an object unrelated to the task; the agent follows the incorrect plan rigidly to budget exhaustion. *Example*: “put a clean mug in coffeemachine.” Memo: “move creditcard 2 to countertop 1”; agent shuttles between `coffeemachine 1` and `sinkbasin 1` for 30+ steps.

Root cause and RL implication. All three modes share the same surface symptom—budget exhaustion—but differ in origin. Category (a) exposes a gap in GRPO credit assignment: under episode-

²The final reported success rate of 67.9% uses the best checkpoint; this analysis samples a representative intermediate checkpoint.

level binary reward, a mid-episode DT call that *breaks a loop* and one that *does nothing* receive identical gradient signal, because both failed episodes yield final reward zero. The policy therefore converges to a safe local optimum: front-load deliberation at step 1 and suppress re-invocation thereafter.

A targeted fix is *sub-sequence advantage estimation for DT tokens*: the advantage for a `deep_think` call at step t is computed from $\sum_{t'=t}^T r_{t'}$ rather than the episode total, so a loop-breaking DT call receives high positive advantage while a redundant call receives near-zero—directly incentivising deliberation at the moment of deadlock without changing the reward function or rollout procedure. This fix addresses category (a); categories (b) and (c) likely require an additional quality signal for DT outputs (e.g., preference-based supervision on memo content), which we leave to future work.

G Reward Sensitivity Analysis

We reran the Qwen3-8B-RL Sel.-Action experiment with the full 150-step training budget while varying one reward hyperparameter at a time and keeping all other settings fixed. Across the tested ranges, the best setting is the default ($\mu_{\text{init}}^* = 5, \text{SR} = 0.6$), while all variants remain close in success rate, supporting robustness to reasonable choices.

Sweep	Value	ALFWorld SR (%)	Tok	Steps
μ_{init}^*	4	67.2	2269	27.2
	5 (default)	67.9	2272	26.6
	6	65.7	2291	27.9
SR switch	0.5	65.0	2341	28.1
	0.6 (default)	67.9	2272	26.6
	0.7	66.4	2364	27.7

Table 2: Full 150-step reward sensitivity for Qwen3-8B-RL Sel.-Action on ALFWorld (134 episodes). Each row uses the same seed, training budget, and evaluation protocol as the main RL result; only the indicated initialization or switch threshold changes.

The initial-reference sweep changes success rate by at most 2.2 points and the switch-point sweep by at most 2.9 points. The default setting therefore lies near the best observed point without requiring a narrow hyperparameter choice.

H Full Results Across All Backbones

Model	Mode	ALFWorld (134 ep.)			ToolHop (995 q.)				GAIA (103 it.)	
		SR %	Tok	Steps	SR %	Tok	Proc. %	Steps	SR %	Tok
GPT-5.2	ReAct	78.4	2008	23.6	55.2	727	55.1	8.0	41.7	16788
	Dyn.-Replan	75.4	1200	–	63.8	646	–	–	36.9	3000
	Pre-Act	70.1	1700	–	65.8	681	–	–	36.9	3500
	Reflexion	77.6	2204	25.4	57.7	885	57.9	7.8	41.7	16119
	Always	84.3	4068	17.0	66.4	2212	66.9	6.6	44.7	34612
	Periodic	<u>89.6</u>	2454	20.4	62.1	878	62.4	6.8	46.6	18469
	Sel.-Gate	82.8	1074	23.7	58.1	795	57.2	8.0	45.6	20265
	Sel.-Action	91.0	1460	19.7	<u>65.1</u>	760	65.3	7.0	46.6	16926
DeepSeek-V4-Pro Preview	ReAct	73.1	2009	25.7	65.1	615	64.9	7.5	30.1	4270
	Reflexion	70.9	1206	25.2	68.5	621	68.4	7.5	34.0	6605
	Always	79.9	2624	25.1	<u>69.1</u>	1970	69.2	7.4	35.0	8913
	Dyn.-Replan	<u>82.8</u>	2173	24.9	68.9	914	67.2	7.8	<u>35.9</u>	4495
	Pre-Act	76.1	2355	26.2	68.4	914	66.1	7.3	32.0	5368
	Periodic	78.4	1335	25.4	67.8	883	68.3	8.2	36.9	4775
	Sel.-Gate	79.9	1669	24.9	62.4	1542	62.0	8.0	34.0	7114
	Sel.-Action	83.6	1335	24.5	69.8	1020	69.8	7.5	<u>35.9</u>	4039
GPT-4o	ReAct	61.2	1678	30.5	59.8	716	60.3	8.3	34.0	8651
	Dyn.-Replan	58.9	2300	–	55.7	672	–	–	30.0	3800
	Pre-Act	54.5	4200	–	56.2	689	–	–	28.2	4100
	Reflexion	64.9	1545	28.9	61.4	598	60.5	8.5	35.0	7618
	Always	64.9	4239	29.4	65.6	3207	65.0	8.7	<u>38.8</u>	21612
	Periodic	<u>73.1</u>	2074	27.9	<u>67.1</u>	621	67.5	8.8	36.9	12646
	Sel.-Gate	67.2	2147	30.1	59.7	1005	58.1	8.3	35.9	8516
	Sel.-Action	77.6	1606	27.6	69.1	627	68.1	7.8	40.8	7384
Qwen3-32B	ReAct	61.2	4715	32.6	52.4	2219	49.6	6.0	22.3	3953
	Pre-Act	77.6	7100	–	55.0	2800	–	–	17.9	3200
	Dyn.-Replan	<u>73.9</u>	7100	–	58.4	2500	–	–	17.0	3400
	Reflexion	62.6	7466	28.5	49.8	2328	47.9	6.6	23.3	3827
	Always	62.6	12613	28.1	36.9	3573	29.4	4.8	25.2	5287
	Periodic	66.4	8928	26.4	<u>56.4</u>	2470	52.2	5.6	27.2	4154
	Sel.-Gate	67.2	4865	26.1	55.0	2608	54.7	5.8	<u>26.2</u>	3807
	Sel.-Action	63.4	5502	31.2	51.7	2822	52.0	6.5	22.3	3140
Llama-3.3-70B-Instruct	ReAct	49.3	5269	33.2	54.4	537	54.8	5.9	13.6	903
	Dyn.-Replan	49.3	4200	–	55.7	697	–	–	15.1	1300
	Pre-Act	38.8	2500	–	58.4	1000	–	–	16.0	1000
	Reflexion	49.3	5493	34.8	54.9	678	55.7	6.1	21.4	1201
	Always	53.0	10271	33.4	53.4	1810	52.0	5.9	<u>21.4</u>	2925
	Periodic	64.2	6326	30.5	57.2	1233	58.2	6.6	22.3	1615
	Sel.-Gate	<u>60.4</u>	4949	33.1	<u>57.7</u>	926	58.1	6.1	22.3	1164
	Sel.-Action	50.0	4661	38.0	53.2	932	52.9	7.7	18.4	1280
Qwen3-8B	ReAct	46.3	3107	32.5	39.6	628	38.1	7.2	12.6	3676
	Reflexion	45.5	3358	32.4	35.7	540	35.6	8.0	15.5	4770
	Always	49.3	5210	33.1	43.5	1171	43.5	6.9	14.6	7128
	Dyn.-Replan	50.7	2414	32.0	43.1	1092	46.9	9.5	12.8	4168
	Pre-Act	44.0	3241	35.6	44.1	790	47.7	8.9	8.8	3801
	Periodic	53.0	3505	29.5	<u>44.3</u>	984	44.9	6.7	<u>16.5</u>	6532
	Sel.-Gate	<u>52.2</u>	2979	30.6	45.1	1015	45.4	7.6	18.4	3376
	Sel.-Action	45.5	3297	35.5	42.6	793	42.1	7.0	14.6	3890
Qwen3-8B-RL Sel.-Action		67.9	2272	26.6	48.2	764	48.4	6.5	21.4	4216
Qwen3-8B-RL Sel.-Gate		61.9	2667	28.1	<u>47.7</u>	989	46.9	6.7	<u>20.4</u>	3316
Qwen3-8B-RL Gate-only		54.5	2871	30.4	<u>45.6</u>	1064	45.8	7.4	17.4	3572
Qwen3-8B-RL ReAct		55.2	1468	29.8	29.7	491	27.5	9.3	9.7	2612

Table 3: Per-benchmark results for all backbones. **Bold / underline** = best / second best per backbone. **Blue** : Selective variants. The last two RL rows are attribution ablations: *ReAct* trains the plain ReAct policy under the identical reward with no separate *deep_think* channel, and *RL-Gate-only* learns the binary gate while freezing the downstream Reason+Action model. Tok = agent output tokens; Proc. = ToolHop process-quality score (%). For newly added planning baselines, unreported Steps and Proc. values are shown as “–” rather than estimated.