From History to State: Constant-Context Skill Learning for LLM Agents

Haoyang Xie, Xinyuan Wang, Yancheng Wang, Puda Zhao, Feng Ju*

School of Computing and Augmented Intelligence, Arizona State University
{hxie40, xwang735, ywan1053, pzhao34, fengju}@asu.edu

Abstract

Large language model (LLM) agents are increasingly used to operate browsers, files, code and tools, making personal assistants a natural deployment target. Yet personal agents face a privacy-cost-capability tension: cloud models execute multi-step workflows well but expose sensitive intermediate context to external APIs, while local models preserve privacy but remain less reliable. Both settings also pay repeatedly for long skill prompts and growing histories. We propose constant-context skill learning, a context-to-weights framework for recurring agent workflows: reusable procedures are learned in lightweight task-family modules, while inference conditions only on the current observation and a compact state block. A deterministic tracker renders this state block from task progress and supplies aligned subgoal rewards, so each module can be trained with step-level SFT and refined through online RL. Across ALFWorld, WebShop, and SciWorld, our agents achieve strong performance across Qwen3-4B, Qwen3-8B, Qwen3-14B, and Llama-3.1-8B. With Qwen3-8B, SFT+RL reaches 89.6% unseen success on ALFWorld, 76.8% success on WebShop, and 66.4% unseen success on SciWorld. They match or exceed strong published agent-training results while reducing prompt tokens per turn by 2–7$\times$ relative to controlled ReAct prompting baselines, showing that procedural context can be moved from prompts into weights. Code is available on GitHub.

1 Introduction

Large language model (LLM) agents are moving from conversational assistants toward systems that execute real multi-turn tasks across browsers, files, code, GUIs, and other tools [13, 41, 42, 26, 25, 40]. As agents become capable of operating tools and files, a natural next step is to deploy them as personal assistants. Systems such as OpenClaw illustrate this direction, where agents interact with personal devices, local tools, and user feedback during normal use [24]. Such local and user-specific deployment makes privacy and cost first-order concerns. Today, the strongest agent models are often accessed through external APIs [23, 3], while models that can run locally are typically less reliable on long-horizon interactive tasks [34]. As a result, personal agents face an uncomfortable trade-off. Cloud models provide stronger execution, but personal agents may handle sensitive data such as emails, calendars, code, documents, and tool outputs, making it undesirable to send every intermediate state to an external service. At the same time, token costs grow quickly with daily use when every step must process long skill descriptions and accumulated execution histories.

Current approaches still treat agent behavior largely as a context-management problem. ReAct-style agents preserve task progress by replaying prior actions, observations, and reasoning traces in the prompt [42]. Memory-augmented agents reduce this burden by storing, organizing, or retrieving past interactions [47, 39, 31, 4]. Recent agent-training methods improve the policy through expert

*Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

trajectories, self-generated experience, or multi-turn preference and reinforcement-learning objectives [28, 6, 32, 5, 44, 20]. These directions substantially improve long-horizon behavior, but they largely preserve the same inference-time interface: before each action, the agent is still prompted with task instructions, skill descriptions, retrieved memories, demonstrations, or accumulated histories. For recurring personal workflows, this is inefficient: the agent repeatedly re-reads procedures it has already practiced, instead of using that experience as part of the policy itself.

A recurring workflow should not require the agent to reread the same procedure, instructions, and examples at every execution. In personal settings, successful executions naturally accumulate as users repeat the same workflow, and even tens of expert trajectories provide many state-action decisions for supervision. This makes the task family a natural unit of learning. Rather than treating repeated executions as isolated episodes, we group variants of the same skill when they share a tool environment, action space, and procedural subgoals. Learning at this level encourages the module to capture the shared procedure across variants, rather than replaying a fixed action sequence from the training trajectories. The prompt only needs the information that changes from step to step, including the current observation and a compact state block describing progress, relevant entities, and remaining subgoals. Inference is therefore shorter but still well specified. The agent need not reread long skill descriptions or replayed histories, yet it has enough state to choose the next action. We keep the base model fixed and learn a separate lightweight skill module for each task family, first from expert demonstrations and then through interaction. Adding a new workflow only requires training a new module. At inference time, running a workflow loads its corresponding module, keeping computation modest and avoiding catastrophic forgetting across unrelated skills.

Our solution: We instantiate this idea as a context-to-weights training pipeline. For each task family, we train a lightweight skill module with step-level supervised fine-tuning (SFT) and then refine it with interaction-based reinforcement learning (RL). A deterministic task tracker provides the interface between raw interaction and learning. It summarizes control-relevant progress into a compact state block, such as acquired objects, completed subgoals, and remaining requirements, and also supplies shaped subgoal rewards during RL. The tracker is a deterministic algorithm, not an LLM reasoner. It converts observations and actions into task progress through simple environment-specific rules. At inference time, the agent loads the corresponding module and conditions on the current observation and state block, rather than the full execution history or long skill prompt. We evaluate this design on ALFWorld, WebShop, and SciWorld [30, 41, 36], which provide representative recurring workflows analogous to personal-agent use cases, including household manipulation, web interaction and procedural tool use.

Our experiments show that this context-to-weights design improves both task performance and context efficiency. With Qwen3-8B, SFT+RL reaches 83.6% / 89.6% success on ALFWorld seen/unseen, 76.8% success on WebShop, and 62.9% / 66.4% success on SciWorld seen/unseen. At the same time, it substantially reduces inference context. Compared with full-history ReAct, our method reduces prompt tokens per turn from 1.3k to 0.18k on ALFWorld, from 3.1k to 0.49k on WebShop, and from about 2.0k to 0.49k on SciWorld. It also reduces total tokens per episode from 34k to about 3k on ALFWorld, from 47k to 3.4k on WebShop, and from 42k to 18k on SciWorld. Building on these findings, we summarize our main contributions as follows:

- We formulate recurring personal-agent workflows as *constant-context skill learning*, where reusable procedures are moved from prompts and growing histories into lightweight skill-module weights, while step-specific progress remains explicit in a compact state block.
- We introduce a modular SFT+RL training pipeline that learns lightweight skill modules from expert trajectories and refines them through interaction, pairing deterministic state-block construction with subgoal rewards aligned to the same tracker fields.
- We conduct controlled evaluations, ablations, and context-efficiency analyses across ALFWorld, WebShop, and SciWorld using Qwen and Llama backbones, showing that bounded-context skill modules can match strong agent-training baselines while reducing prompt-token cost by 2–7$\times$ relative to ReAct baselines.

2 Problem Setup

Task families. We study recurring agent workflows organized as task families. A task family k is a set of episode variants that share a tool environment, action space, and procedural structure, while

differing in episode-specific goals, objects, constraints, or initial states. In personal-agent use, such families arise from repeated workflows, such as ordering items through the same interface, organizing files with the same toolchain, or running variants of the same analysis procedure. In benchmarks, they correspond to predefined families such as ALFWorld object manipulation, WebShop shopping, and SciWorld science-lab tasks. This is a natural unit of learning because variants share reusable procedure but still require episode-specific decisions. At the start of an episode, the agent receives an instruction $g \sim \mathcal{G}_k$. At step t, it observes o_t, produces textual action a_t, and continues until termination, yielding $\tau = (g, o_1, a_1, \ldots, o_T, a_T)$. For each family, we assume a modest set of successful executions $\mathcal{T}_k$, from user demonstrations or prior successful agent runs. We use the task-family definitions provided by each benchmark. Specifically, these are ALFWorld's six official task types, WebShop's shared shopping workflow, and SciWorld's ETO grouping. We use five ALFWorld adapters because Pick2 reuses the Pick adapter twice.

Standard history-based formulation. Most prompting-based agents maintain task progress by placing more information into the context. Let d_k denote task-family context such as skill instructions, demonstrations, retrieved memories, or examples, and let $h_t = (o_1, a_1, \ldots, o_t)$ be the execution history. A standard history-based agent predicts actions from $c_t^{\text{hist}} = \text{Format}(g, d_k, h_t)$. This formulation covers ReAct-style prompting, retrieved-memory agents, and skill-prompted agents: they differ in how d_k or h_t is assembled, but the next action is still generated from a text context that contains reusable procedure and accumulated interaction state. Its length scales as

$$|c_t^{\text{hist}}| \approx |g| + |d_k| + \sum_{i=1}^{t} (|o_i| + |a_i|), \tag{1}$$

This is inefficient for recurring workflows. d_k is reprocessed at every step, while the history term grows with the episode length and can make relevant state harder to use reliably [17].

Bounded-context skill learning. We instead seek a task-family policy whose input remains bounded as the episode grows. For each family k, we keep a frozen base model θ_0 and learn a lightweight skill module ϕ_k. At step t, the model conditions on $x_t = \text{Format}(g, o_t, q_t, b_t)$, where $q_t = (o_{t-1}, a_{t-1})$ is one-step context and b_t is a compact state block. The state block contains progress information, such as the current subgoal, selected entities, acquired objects, chosen options, checked locations, or remaining requirements. The desired policy is

$$a_t \sim \pi_{\theta_0, \phi_k}(\cdot \mid x_t), \qquad |x_t| \leq B_k \quad \forall t, \tag{2}$$

where B_k does not scale with the trajectory length. Thus, ϕ_k represents the reusable procedure, while x_t carries only step-specific state. This bounded-context objective creates a state challenge that the agent must avoid carrying the full history, but it also cannot simply discard history altogether. The current observation alone may omit important progress information, such as selected entities or completed subgoals, and different tasks require different pieces of past state. We use b_t to denote the compact state block that should carry this information. Its role is not to restate the whole trajectory or provide a solution plan, but to expose the minimal state variables needed for the next decision.

3 Context-to-Weights Skill Learning

Given the bounded-context formulation in Section 2, we now instantiate it as a concrete context-to-weights training pipeline. The pipeline has three stages (Figure 1). First, a deterministic task tracker compresses the relevant interaction state into a compact state block. Second, successful executions are converted into step-level supervision for training a task-family skill module. Third, the module is refined through interaction using rewards derived from the same subgoal structure. Throughout training and inference, the base model θ_0 remains frozen, only the task-family module ϕ_k is updated.

3.1 From History to State Blocks

The purpose of the state block is to preserve decision-relevant progress without passing the full execution history to the model. For each task family k, we define a deterministic tracker $\mathcal{M}_k$ that maintains a structured state m_t [8, 46]. Initialized from the task instruction g, m_t stores variables from which the state block can be rendered, including task targets, acquired or selected entities, checked locations, completed subgoals, and other family-specific progress indicators. Within a task

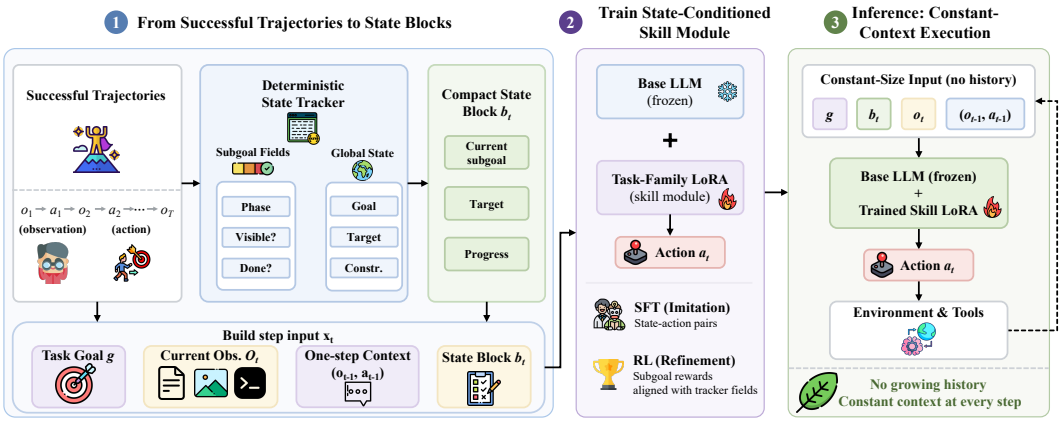

Figure 1: Context-to-weights skill learning pipeline.

family, an episode progresses through procedural phases or subgoals, such as search, manipulation, selection, or placement. The tracker records the current phase as part of m_t. At step t, the tracker updates its state as $m_t = \mathrm{Update}_k(m_{t-1}, a_{t-1}, o_t)$ and renders the model-facing state block as $b_t = \mathrm{Render}_k(m_t)$. Algorithm 1 gives the generic update-and-render interface.

The tracker is deterministic rather than an LLM summarizer, so its outputs are reproducible and require no additional model calls. It updates task-family variables with lightweight parsing rules, making state construction inexpensive. The rendered block records only state facts needed for the next decision, not a trajectory summary or full solution plan.

To reduce manual engineering when extending the framework to new workflows, we also investigate whether an LLM can generate the state schema, tracker update logic, and reward rules offline. Appendix 7.5 provides a representative reward-drafting prompt. We evaluate the generated tracker and reward pipeline on WebShop in Section 4.6, with generation and validation details in Appendix 7.8. The generated tracker runs deterministically without LLM calls during training or inference.

The tracked variables depend on the family and phase of the task. In ALFWorld, the state block records fields such as the target object, whether the agent is holding it, the destination receptacle, and checked locations. In WebShop, it tracks the current query, inspected product, selected options, remaining options, and purchase readiness. In SciWorld, it tracks the current phase, selected entity, inventory, answer room, and completed subgoals. Across these settings, the same interface is used: the policy receives the current observation o_t, one-step context $q_t = (o_{t-1}, a_{t-1})$, and the rendered state block b_t:

$$x_t = \mathrm{Format}(g, o_t, q_t, b_t). \tag{3}$$

Thus, past interactions affect the policy only through the tracker state and one-step context, rather than through a growing history window.

Algorithm 1 Tracker Update and Render

Require: $k, g, m_{t-1}, a_{t-1}, o_t$
Ensure: m_t, b_t
$z_t \leftarrow \mathrm{Parse}_k(g, o_t, a_{t-1})$
entities, locations, options, outcomes
$m_t \leftarrow m_{t-1}$
Update global fields in m_t from z_t
targets, constraints, inventory
Update phase-specific fields in m_t from z_t
checked locations, selected options, results
$b_t \leftarrow \mathrm{Render}_k(m_t)$
render fields relevant to current phase/subgoal
return m_t, b_t

Algorithm 2 State-Action Samples

Require: $k, \mathcal{T}_k, \mathcal{M}_k$
Ensure: $\mathcal{D}_k$
$\mathcal{D}_k \leftarrow \emptyset$
for each $\tau = (g, o_1, a_1^\star, \ldots, o_T, a_T^\star) \in \mathcal{T}_k$ **do**
  $m_0 \leftarrow \mathcal{M}_k.\mathrm{Init}(g)$; $o_0, a_0^\star \leftarrow \varnothing$
  for $t = 1, \ldots, T$ **do**
    $m_t \leftarrow \mathcal{M}_k.\mathrm{Update}(m_{t-1}, a_{t-1}^\star, o_t)$
    $b_t \leftarrow \mathcal{M}_k.\mathrm{Render}(m_t)$
    $x_t \leftarrow \mathrm{Format}(g, o_t, (o_{t-1}, a_{t-1}^\star), b_t)$
    $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \{(x_t, a_t^\star)\}$
  end for
end for
return $\mathcal{D}_k$

3.2 Training Task-Family Skill Modules

For each task family k, we replay successful executions from $\mathcal{T}_k$ through the tracker and collect step-level supervision pairs as detailed in Algorithm 2, where $a_t^\star$ denotes the expert action from the successful trajectory. Algorithm 2 invokes the update-and-render procedure in Algorithm 1 at each step. This yields

$$\mathcal{D}_k = \{(x_t, a_t^\star) : \tau \in \mathcal{T}_k,\ t = 1, \ldots, T_\tau\}. \tag{4}$$

The module is trained with standard next-action supervision:

$$\mathcal{L}_{\mathrm{SFT}}^{(k)} = - \sum_{(x_t, a_t^\star) \in \mathcal{D}_k} \log p_{\theta_0, \phi_k}(a_t^\star \mid x_t). \tag{5}$$

The target is the executable environment action, not a reasoning trace or a choice among enumerated valid actions. The policy therefore learns to generate the next action text directly from x_t, rather than relying on a per-step action list supplied in the prompt. In our implementation, each task-family module ϕ_k is one LoRA adapter [10]. LoRA keeps each pretrained matrix W_0 fixed and learns a low-rank update:

$$W_k = W_0 + \Delta W_k, \qquad \Delta W_k = \frac{\alpha}{r} B_k^{\mathrm{LoRA}} A_k^{\mathrm{LoRA}}, \tag{6}$$

where $A_k^{\mathrm{LoRA}} \in \mathbb{R}^{r \times d_{\mathrm{in}}}$, $B_k^{\mathrm{LoRA}} \in \mathbb{R}^{d_{\mathrm{out}} \times r}$, and $r \ll \min(d_{\mathrm{in}}, d_{\mathrm{out}})$. The superscript distinguishes these LoRA matrices from the context bound B_k in Eq. 2. Only these adapter parameters are trained, while the shared base model remains frozen. This gives each task family its own lightweight skill module without overwriting unrelated skills [9, 7, 38].

This stage transfers repeated procedural knowledge from prompt-time context into task-family parameters. The module learns how to act from the compact state representation across many variants of the same workflow, while episode-specific information remains explicit in x_t.

3.3 Refining Skill Modules with Subgoal-Guided RL

Supervised training imitates successful trajectories, but online execution can lead the learned policy into off-trajectory states. We therefore refine each skill module through online interaction. During RL, rollouts use the same bounded input x_t as in supervised training: after each action, the environment returns a new observation, the tracker updates m_t, and the next state block b_t is rendered.

The key design is that the same tracker state m_t used to render b_t also supplies subgoal-level reward signals, so RL reinforces progress on the variables exposed in the state block. At each step, the reward combines the benchmark environment signal with deterministic progress and error terms:

$$r_t = r_t^{\mathrm{env}} + r_t^{\mathrm{prog}} - r_t^{\mathrm{err}}. \tag{7}$$

Here r_t^{env} is the environment score or success signal. The progress and error terms, r_t^{prog} and r_t^{err}, are defined around the same subgoals and state-block fields used by the tracker, following the broader idea of shaping sparse task rewards with intermediate progress signals [22]. As a result, RL encourages the policy to use the state block as its execution state. The progress term r_t^{prog} rewards actions that advance the current subgoal, while the error term r_t^{err} penalizes invalid, repetitive, or phase-inconsistent actions, together with a small step cost. We use a strong reasoning model (GPT-5.5 [23]) offline to help design these rules from the task specification and the state-block schema. The rewards are then fully deterministic at rollout time, with no LLM judge queried during RL. Appendix 7.2 gives the reward specification and Appendix 7.5 shows the prompt used to ask the reasoning model to draft deterministic reward rules from the state-block schema.

We use a GRPO-style group-normalized policy-gradient update [27]. For each task instance, the current policy samples a group of K rollouts, indexed by i. Let T_i be the length of rollout i, and let $r_u^{(i)}$ be its shaped reward at step u. We compute the discounted step return $G_t^{(i)} = \sum_{u=t}^{T_i} \gamma^{u-t} r_u^{(i)}$, then normalize these returns across all steps from the K rollouts of the same task instance:

$$A_t^{(i)} = \frac{G_t^{(i)} - \mu_{\mathrm{group}}}{\sigma_{\mathrm{group}} + \epsilon}, \qquad \mu_{\mathrm{group}}, \sigma_{\mathrm{group}} = \mathrm{MeanStd}\left(\{G_t^{(i)}\}_{i,t}\right). \tag{8}$$

Here γ is the discount factor and ϵ is a small constant for numerical stability. This compares attempts under the same task difficulty and avoids training a separate value function.

Table 1: Main task performance (mean±std over inference seeds; **bold** = best per column). Score is benchmark-provided. SR denotes success rate, measured as binary full-task success (%).

Base Model	Method	ALFWorld		WebShop		SciWorld			
		SR				Seen		Unseen	
		Seen	Unseen	Score	SR	Score	SR	Score	SR
Qwen3-4B	Ours: SFT only	59.5±3.5	56.7±3.3	66.4±1.4	59.6±2.3	67.8±1.9	59.3±1.7	61.0±1.0	52.0±0.3
	Ours: SFT + RL	76.4±1.5	81.3±1.9	86.0±0.6	76.1±1.1	72.4±1.9	63.1±1.5	75.1±1.4	65.4±1.4
Qwen3-8B	Ours: SFT only	62.4±2.5	61.7±2.2	69.5±1.7	62.2±1.2	65.7±1.6	58.2±1.5	61.5±0.9	52.0±0.7
	Ours: SFT + RL	**83.6±1.8**	**89.6±0.4**	84.0±0.0	76.8±0.2	72.8±2.7	62.9±2.2	**79.7±0.2**	66.4±0.5
Llama-3.1-8B	Ours: SFT only	61.0±3.4	60.2±1.7	69.1±1.7	60.3±1.0	66.5±1.5	61.0±1.6	57.6±0.7	50.6±1.0
	Ours: SFT + RL	73.6±0.7	80.6±2.6	**86.3±1.5**	75.7±1.3	74.1±1.0	66.2±2.1	73.9±0.6	64.1±0.3
Qwen3-14B	Ours: SFT only	62.9±2.7	61.4±2.3	66.5±1.5	59.8±1.8	68.9±1.4	61.9±1.3	61.6±0.6	53.1±0.8
	Ours: SFT + RL	82.1±0.8	87.3±1.6	85.9±0.5	**78.8±0.8**	**76.0±1.6**	**67.0±1.8**	79.1±0.1	**67.3±0.3**

The sampled action at step t of rollout i is $a_t^{(i)}$, generated from the bounded input $x_t^{(i)}$. We update only the task-family module with a sampled policy-gradient loss and a reference-policy penalty to the frozen SFT adapter:

$$\mathcal{L}_{\mathrm{RL}}^{(k)} = \mathbb{E}_{i,t}\left[-A_t^{(i)} \log \pi_{\theta_0,\phi_k}(a_t^{(i)} \mid x_t^{(i)}) + \beta\left(\log \pi_{\theta_0,\phi_k}(a_t^{(i)} \mid x_t^{(i)}) - \log \pi_{\theta_0,\phi_k^{\mathrm{SFT}}}(a_t^{(i)} \mid x_t^{(i)})\right)\right]. \tag{9}$$

Here ϕ_k^{SFT} is the frozen SFT-stage adapter, which keeps RL close to the action format and procedural behavior learned during supervised training. During refinement, θ_0 remains frozen and only ϕ_k is updated.

4 Experiments and Results

We evaluate the proposed context-to-weights pipeline on three representative agent benchmarks, using Qwen and Llama backbones at different model scales. Our experiments are organized around four research questions. (**RQ1**) Does task-family skill learning improve agent performance across benchmarks and backbones, and how much does interaction-based RL refinement add over SFT-only skill modules? (**RQ2**) How much inference context does the bounded-context interface save compared with ReAct-style prompting, a representative history-based agent interface that repeatedly conditions on long instructions and growing execution histories? (**RQ3**) Which components of the pipeline, including the state block, one-step context, SFT training, and subgoal-guided reward shaping, are responsible for the gains? (**RQ4**) How does the method compare with prior agent methods across memory-based prompting, learned memory and RL-based agent training? We further examine robustness across task variants and the feasibility of LLM-generated trackers and reward rules.

4.1 Experimental Setup

Benchmarks. We evaluate on three text-mode agent benchmarks that approximate recurring personal-agent workflows in different fixed environments. **ALFWorld** [30] represents household assistance: the agent navigates simulated rooms, finds target objects, applies transformations such as cleaning, heating, or cooling and places objects in target receptacles. We evaluate on 140 seen and 134 unseen games. **WebShop** [41] represents shopping assistance: the agent searches for products, opens item pages, selects required options, and purchases the target item. We use the 1,000-product text setting and evaluate on 500 test goals. **SciWorld** [36] represents procedural tool use: the agent manipulates objects, uses instruments, performs experiments, and reports answers in science-lab environments. We evaluate on 194 seen and 211 unseen variants following the ETO split structure [32]. All three use the same agent interface, text observations in and executable text actions out. Figures 2, 5, and 6 show representative interactions from these settings. Additional details on task families, training trajectories, and evaluation settings are provided in Appendix 7.1.

Baselines. Our baselines serve two purposes. For a *controlled token-cost comparison*, we run two ReAct variants under the same evaluation setup as our method: **ReAct-1step** retains only the

Table 2: Context and compute efficiency with Qwen3-8B under controlled prompting comparisons. All methods use the same tokenizer, environment split, and action parser.

Metric	Method	ALFWorld		WebShop	SciWorld	
		Seen	Unseen	Test	Seen	Unseen
Avg. Steps	ReAct, 1-step history	24.9	25.3	13.6	18.9	12.3
	ReAct, full history	21.6	21.8	14.4	13.4	12.5
	Ours: SFT + RL	18.4	15.9	6.8	33.7	30.7
Prompt Tok./Turn	ReAct, 1-step history	380.0	358.6	1,059.0	1,443.0	1,481.0
	ReAct, full history	1,310.5	1,320.4	3,092.7	2,038.9	1,938.0
	Ours: SFT + RL	183.8	179.9	488.0	492.0	496.3
Completion Tok./Turn	ReAct, 1-step history	53.3	52.3	72.4	61.9	65.3
	ReAct, full history	50.9	52.9	71.4	54.6	54.3
	Ours: SFT + RL	7.1	7.1	10.4	5.3	5.3
Total Tok./Episode	ReAct, 1-step history	10,969	10,443	15,763	28,088	18,953
	ReAct, full history	34,199	34,145	46,531	42,280	39,550
	Ours: SFT + RL	3,251	2,851	3,353	17,894	16,182

most recent observation and action, while **ReAct-full** appends the available trajectory history with truncation when needed. Prior prompt- and memory-based agents differ in their prompts, retrieved content, and history policies, but they usually include at least recent interaction context and often much more. The two ReAct variants therefore represent the main short-context and history-based settings used by prompting-based agents, giving a controlled efficiency comparison in Table 2. For *performance comparison*, we report published state-of-the-art agent-training results in Table 6.

Implementation details. We use Qwen3-8B as the default backbone and repeat the main experiments with Qwen3-4B, Qwen3-14B, and Llama-3.1-8B to test robustness across model scale and family. For each task family, the base model is frozen and only a LoRA skill module is trained, updating about 2% of the parameters, 0.5–0.7 GB on disk per adapter. We first train modules with SFT and then refine them with online GRPO-style RL using four rollouts per task instance and the subgoal rewards. All main training runs fit on a single NVIDIA A100 80GB GPU. Main performance is reported as mean$\pm$std over three inference seeds with temperature 0.4 and top-p 0.95, while token-cost results use deterministic decoding for stable accounting. Full hyperparameters, decoding settings, and compute details are reported in Appendix 7.1.



Figure 2: Qualitative ALFWorld household trace with rendered state blocks.

4.2 Main Task Performance

Table 1 shows that moving recurring procedures from context into task-family modules is effective across benchmarks. SFT already gives strong performance, indicating that expert trajectories provide enough step-level supervision for the model to act from compact per-step inputs rather than long skill prompts or full histories. On WebShop, for example, the untrained prompt variant with a state

block reaches only 23.6% success in Table 3, while SFT raises success to 62.2% with Qwen3-8B. RL refinement further improves the trained modules. With Qwen3-8B, SFT+RL reaches 83.6% / 89.6% success on ALFWorld seen/unseen, 76.8% success on WebShop, and 62.9% / 66.4% success on SciWorld seen/unseen. Across the four backbones, RL improves over SFT by roughly 13–28 percentage points on ALFWorld, 15–19 points on WebShop, and 5–15 points on SciWorld. The improvements are also not tied to a single backbone. Qwen3-8B gives the strongest ALFWorld performance and the best SciWorld unseen score, while Llama-3.1-8B is strongest on WebShop score. Qwen3-14B reaches 78.8% WebShop success and 67.0% / 67.3% SciWorld seen/unseen success, confirming that the training gains persist at the larger scale. Qwen3-4B also benefits from the same training recipe despite its smaller size. This pattern suggests that the gains come from the task-family training setup, not from a particular pretrained model. As a modularity check, we also train a single unified Qwen3-8B module across all ALFWorld families, which reaches 74.3% / 70.9% seen/unseen success after SFT and 86.4% / 88.1% after RL. This shows that a unified module is viable, but we use task-family modules for realistic deployment because they reduce interference among different skills, especially for smaller local models, and avoid training bias from imbalanced task-family data. A cross-benchmark SFT experiment further shows that a single adapter can support all three environments, with some ALFWorld unseen interference (Appendix 7.9). We analyze the individual contributions of the state block, SFT, and RL in Section 4.4.

4.3 Context and Token Efficiency

The efficiency measurements below concern inference-time tokens. Task quality is reported alongside the training stages in Table 1, while Table 6 includes the controlled ReAct baselines. The token reductions do not include the one-time offline training cost, reported separately in Appendix 7.1.

Table 2 shows that the bounded-context interface substantially reduces inference context. ReAct-1step is already a strong efficiency baseline because it keeps only the latest observation and action. Many prompt- and memory-based agents keep more history, retrieved memories, or both. Our method still uses 2.1–3.0$\times$ fewer prompt tokens per turn than ReAct-1step on all three benchmarks and reduces prompt tokens by about 4–7$\times$ relative to ReAct-full. This saving comes from moving reusable procedural context into the learned module, not merely from dropping history.

The episode-level totals show the same pattern. On ALFWorld and WebShop, failed executions often continue producing unproductive actions until the step limit, so better execution also reduces average steps. SciWorld behaves differently, invalid actions can terminate an episode early with negative score, while successful procedures can be long, with task-specific limits up to 100 steps. Thus, stronger SciWorld agents may take more steps because they survive and complete more procedures. Even there, our method substantially reduces total token use. Relative to ReAct-full, total token use drops by about 10–12$\times$ on ALFWorld, 14$\times$ on WebShop, and 2–3$\times$ on SciWorld.

4.4 Component Ablations

Table 3: Ablation study on WebShop with Qwen3-8B.

Method	Score	SR
Prompt design (no training):		
Current observation only	10.1	1.2%
+ one-step context	34.2	5.6%
+ subgoal state block	42.6	23.6%
Trained leave-one-out variants:		
w/o state block	72.9	61.2%
w/o reward shaping	83.1	72.2%
Trained methods:		
SFT only	69.5	62.2%
Full method	**84.0**	**76.8%**

We ablate components on WebShop with Qwen3-8B, evaluated on all 500 test goals. This default-backbone,single-module setting makes component changes easy to isolate. Table 3 shows that the state block is the main prompt-side signal. Without training, the current observation alone is near floor performance, and adding one-step context only raises success from 1.2% to 5.6%. Adding the state block raises success to 23.6%, a much larger gain, showing that compact progress state is the key information missing from short prompts.

Training then moves reusable procedure into the skill module. SFT raises success from 23.6% to 62.2%, and RL further improves it to 76.8%. The trained leave-one-out variants are all evaluated after RL refinement. Removing the state block drops success by 15.6 points, confirming that the model still uses it as execution state. Section 4.5 separately tests robustness to changes in instruction wording, goal combinations, and state serialization.

Using only terminal task reward during RL drops success by 4.6 points, showing that subgoal rewards provide useful intermediate feedback. Figure 3 further decomposes the reward design. Progress rewards guide option and subgoal completion, error penalties reduce invalid or repeated actions, and the step cost discourages unnecessary actions. Removing any component lowers success, showing that the shaped reward works as a combined signal. The full reward specification is listed in Appendix 7.2. Appendix 7.3 further studies SFT data efficiency on WebShop.

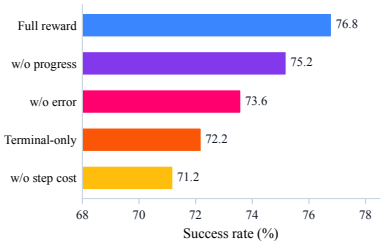

Figure 3: Reward ablation.

4.5 Generalization Across Task Variants

To examine robustness across task variants, we test whether the learned policies remain effective under changes to instruction wording, goal composition, and state serialization. We evaluate fixed Qwen3-8B SFT+RL adapters on paraphrased instructions, unseen WebShop constraint combinations, and JSON state serialization (Table 4). Paraphrases change only model-visible instructions. Tracker and environment goals remain unchanged. Results use three inference seeds (Appendix 7.6). State reserialization and ALFWorld paraphrasing retain near-original performance, while unseen constraint combinations achieve 75.8% SR. This pattern supports robust execution across input variants. WebShop paraphrasing retains 72.2% SR, indicating greater wording sensitivity.

Table 4: Qwen3-8B SFT+RL generalization: SR (%), mean±std over three inference seeds; N goals per seed.

Input condition	N	SR
WebShop: original	500	76.8 ± 0.2
Paraphrased instruction	500	72.2 ± 0.7
Unseen constraint combination	446	75.8 ± 0.6
State as sorted JSON	500	75.7 ± 0.8
ALFWorld unseen: original	134	89.6 ± 0.4
Paraphrased instruction	134	89.1 ± 1.1

4.6 Automating the State Interface

To reduce manual engineering for new workflows, we investigate LLM-based generation of state schemas, trackers, and reward rules. GPT-5.4 generates a WebShop state schema, tracker, renderer, and deterministic reward rules from the public interface and expert training examples, with our tracker and 500 test goals withheld. Replay validation precedes SFT and RL. The generated code requires no runtime LLM calls. SFT with the generated interface reaches 61.2% SR, close to the manual tracker's 62.2%. RL further improves it to 65.8% (Table 5), demonstrating that the generated state and reward rules support both training stages. Appendix 7.8 gives the full protocol and metrics.

Table 5: WebShop SR (%) with Qwen3-8B on 500 goals.

Tracker	SFT	SFT+RL
Manual	62.2	76.8
LLM-generated	61.2	65.8

4.7 Comparison with Published Methods

Table 6: Comparison with published baselines. % denote success rates, otherwise values are task scores.

Method	ALFWorld	WebShop	SciWorld
ReAct-1	23.1%	8.8 / 1.2%	-50.0
ReAct-full	46.3%	7.0 / 3.8%	-21.5
ETO [32]	72.4%	67.4 / 37.5%	65.0
DMPO [28]	88.8%	64.3 / –	65.1
GiGPO [5]	**90.8%**	84.4 / 72.8%	–
MemGen [43]	90.6%	–	–
Early Exp. [44]	85.9%	77.0 / 62.2%	68.0
EMPO2 [20]	–	**88.3 / 76.9%**	75.9
ACT [19]	88.1%	– / 33.8%	50.34
Ours	89.6%	84.0 / 76.8%	**79.7**

Table 6 compares our method with published agent-training baselines. These results are copied from the original papers. We discuss these families of methods in more detail in Section 5. The comparison is not fully controlled, since the methods differ in training objectives, data construction, and implementation details. However, it is still informative: most methods use similar-scale open backbones such as Qwen2.5-7B or Qwen3-8B, and overlapping results are reported under the same settings, including the 1,000-product WebShop setting and the ETO SciWorld split.

Our method is also strong compared with published agent-training results. On ALFWorld, it reaches 89.6% success, only 1.2 points below the strongest listed result. On WebShop, it reaches 76.8% success, essentially tied with the strongest listed result of 76.9%. On SciWorld, it obtains the best listed score, 79.7 versus EMPO2's 75.9. We obtain these results with a

modest training budget, with all training runs using a single A100 GPU, whereas strong published RL baselines such as GiGPO report using multiple H100 GPUs for training. This performance also comes with a shorter inference context. Table 2 reports the measured token savings relative to ReAct-1step and ReAct-full under controlled settings.

5 Related Work

Prompting and memory-augmented agents. ReAct-style agents maintain task progress by placing prior observations, actions, and reasoning traces in the prompt [42]. A large line of memory-augmented agents reduces this burden by storing, summarizing, retrieving, or compressing past interactions [47, 39, 31, 4, 43, 14, 16]. Prompt-compression methods similarly reduce inference cost by shortening the text context [11, 21, 15]. These methods improve long-horizon behavior, but reusable procedural knowledge remains represented as text that must be selected and reprocessed at inference time. Our work takes a complementary direction: recurring procedure is moved into task-family weights, while the prompt carries only the compact information. GEPA evolves prompts through execution-based reflection [1], while ACE maintains an evolving playbook of strategies [45]. Both adapt context rather than task-family weights, complementing our policy-learning approach.

Training language agents. Recent work improves LLM agents with expert trajectories, self-generated experience, preference optimization, or reinforcement learning [29, 28, 49, 6, 32, 5, 44, 20, 19]. These methods substantially improve task performance, and several also incorporate memories or early experiences into training. However, most preserve a prompting-style execution interface, where the agent still conditions on long instructions, histories, retrieved content, or candidate actions. Our pipeline shares the SFT-then-RL training recipe, but trains the policy to act from a bounded-context input consisting of the current observation, one-step context, and a tracker-rendered state block.

Skill modules and subgoal-aligned rewards. Some agents accumulate reusable skills as retrievable text or code libraries [35, 18]. Agent Workflow Memory represents induced workflows as textual guidance [37], while SkillWeaver refines website skills into callable APIs [48]. We share the goal of reusable skills, but move recurring procedures into lightweight task-family weights, avoiding the longer prompts and slower inference caused by repeated skill-text processing. Parameter-efficient adaptation makes this practical without updating the full backbone [10, 7]. New workflows are added by training new adapters, limiting interference and catastrophic forgetting across heterogeneous task families, especially for smaller local models. Group-normalized RL avoids training a separate value model [27]. Our contribution is the feedback design, subgoal rewards are aligned with the tracker fields rendered in the state block, encouraging the module to use compact state during interaction [33, 12, 22].

6 Conclusion and Limitations

We presented constant-context skill learning, a context-to-weights framework for recurring LLM-agent workflows. Instead of repeatedly prompting an agent with long skill descriptions and growing histories, we train lightweight task-family modules that internalize reusable procedure while conditioning on the current observation and a compact tracker-rendered state block. Across ALFWorld, WebShop and SciWorld, the resulting agents achieve competitive task performance while substantially reducing inference context compared with ReAct-style prompting. The results suggest that recurring agent behavior can be made both more capable and more context-efficient by moving procedural context from prompts into weights.

The main limitations concern automation, modality, and transfer. First, the state representation and reward rules are task-family specific. We manually define the tracker fields rendered in the state block, and use a strong reasoning model offline to draft deterministic subgoal rewards from the task description and state-block schema. The LLM-generated WebShop tracker demonstrates that this construction can be automated, but its 65.8% SR trails the mature tracker at 76.8%. This design burden may become easier to automate as agent systems expose `SKILL.md`-style specifications of procedures, tool constraints, and success conditions [2]. Compiling such specifications with execution logs into state-block schemas and aligned subgoal rewards is a promising future direction. Second, our evaluation is limited to text-mode environments. Extending the interface to GUI and multimodal agents is a natural next step. Third, per-family modules do not zero-shot transfer to new skills. Adding a workflow requires training a new adapter, although the base model remains fixed and existing skills are preserved. Moreover, each adapter is only 0.5–0.7 GB on disk and only the active one loads at inference, making the storage cost modest in practice.

References

[1] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pages 8479–8565, 2026.

[2] Anthropic. Agent skills. `https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview`, 2026. Accessed: 2026-05-04.

[3] Anthropic. Claude opus 4.7. `https://www.anthropic.com/news/claude-opus-4-7`, 2026. Accessed: 2026-05-01.

[4] Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. Lightmem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*, 2025.

[5] Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

[6] Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma Gongque, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. Agentrefine: Enhancing agent generalization through refinement tuning. *arXiv preprint arXiv:2501.01702*, 2025.

[7] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

[8] Matthew Henderson, Blaise Thomson, and Steve Young. Word-based dialog state tracking with recurrent neural networks. In *Proceedings of the 15th annual meeting of the special interest group on discourse and dialogue (SIGDIAL)*, pages 292–299, 2014.

[9] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

[10] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *ICLR*, 2022.

[11] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 13358–13376, 2023.

[12] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[13] Yubo Li, Xiaobin Shen, Xinyu Yao, Xueying Ding, Yidi Miao, Ramayya Krishnan, and Rema Padman. Beyond single-turn: A survey on multi-turn interactions with large language models. *arXiv preprint arXiv:2504.04717*, 2025.

[14] Xuechen Liang, Meiling Tao, Yinghui Xia, Jianhui Wang, Kun Li, Yijin Wang, Yangfan He, Jingsong Yang, Tianyu Shi, Yuantao Wang, et al. Sage: Self-evolving agents with reflective and memory-augmented abilities. *Neurocomputing*, 647:130470, 2025.

[15] Barys Liskavets, Maxim Ushakov, Shuvendu Roy, Mark Klibanov, Ali Etemad, and Shane K Luke. Prompt compression with context-aware sentence encoding for fast and improved llm inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24595–24604, 2025.

[16] Junyang Liu, Wenning Hao, Kai Cheng, and Dawei Jin. Large language model-based planning agent with generative memory strengthens performance in textualized world. *Engineering Applications of Artificial Intelligence*, 148:110319, 2025.

[17] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

[18] Shunyu Liu, Yaoru Li, Kongcheng Zhang, Zhenyu Cui, Wenkai Fang, Yuxuan Zheng, Tongya Zheng, and Mingli Song. Odyssey: Empowering minecraft agents with open-world skills. *arXiv preprint arXiv:2407.15325*, 2024.

[19] Weize Liu, Minghui Liu, Sy-Tuyen Ho, Souradip Chakraborty, Xiyao Wang, and Furong Huang. Agentic critical training. *arXiv preprint arXiv:2603.08706*, 2026.

[20] Zeyuan Liu, Jeonghye Kim, Xufang Luo, Dongsheng Li, and Yuqing Yang. Exploratory memory-augmented llm agent via hybrid on-and off-policy optimization. *arXiv preprint arXiv:2602.23008*, 2026.

[21] Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

[22] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, pages 278–287, 1999.

[23] OpenAI. Introducing gpt-5.5. `https://openai.com/index/introducing-gpt-5-5/`, 2026. Accessed: 2026-05-01.

[24] OpenClaw. Openclaw — personal ai assistant. `https://openclaw.ai/`, 2026. Accessed: 2026-05-01.

[25] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

[26] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

[27] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[28] Wentao Shi, Mengqi Yuan, Junkang Wu, Qifan Wang, and Fuli Feng. Direct multi-turn preference optimization for language agents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2312–2324, 2024.

[29] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

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

[31] Yiheng Shu, Saisri Padmaja Jonnalagedda, Xiang Gao, Bernal Jiménez Gutiérrez, Weijian Qi, Kamalika Das, Huan Sun, and Yu Su. Remem: Reasoning with episodic memory in language agent. *arXiv preprint arXiv:2602.13530*, 2026.

[32] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600, 2024.

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

[34] Qwen Team. Qwen3.5-Omni technical report. *arXiv preprint arXiv:2604.15804*, 2026.

[35] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

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

[37] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[38] Lingling Xu, Haoran Xie, S Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

[39] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[40] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[41] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35: 20744–20757, 2022.

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

[43] Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents. *arXiv preprint arXiv:2509.24704*, 2025.

[44] Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, et al. Agent learning via early experience. *arXiv preprint arXiv:2510.08558*, 2025.

[45] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pages 86069–86100, 2026.

[46] Yichi Zhang, Zhijian Ou, Min Hu, and Junlan Feng. A probabilistic end-to-end task-oriented dialog model with latent belief states towards semi-supervised learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9207–9219, 2020.

[47] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

[48] Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

[49] Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. *arXiv preprint arXiv:2402.19446*, 2024.

7 Technical appendices and supplementary material

7.1 Training Details and Hyperparameters

All three benchmarks provide expert trajectories, which we use directly to construct the SFT data after benchmark-specific filtering. Each successful trajectory is converted into step-level bounded-context inputs paired with the expert action, so the number of SFT samples is larger than the number of trajectories. Table 7 reports the resulting data scale and training settings for the main Qwen3-8B experiments. The Qwen3-4B, Qwen3-14B, and Llama-3.1-8B runs use the same recipe unless otherwise specified. Because Qwen3-8B supports a chat-template thinking mode, we disable it during both training and evaluation (`enable_thinking=False`) and use the model as a standard instruction-following language model. Qwen3-4B and Llama-3.1-8B do not have a thinking mode and are used with their default chat templates.

ALFWorld contains five base task families in our training setup: Pick, Clean, Heat, Cool, and Examine. Pick2 is not trained as a separate module because it does not introduce a new procedure: it consists of two sequential executions of the same Pick-and-Place skill. At evaluation time, we therefore reuse the Pick module twice rather than learning an additional Pick2-specific module.

WebShop is treated as a single task family because all examples share the same shopping workflow. To align with standard WebShop evaluation, we use the 1K-product text environment and evaluate on 500 held-out goals.

For SciWorld, we follow the ETO evaluation protocol, which is the common setup used by recent ScienceWorld agent-training work, including the task-family grouping, official train/dev/test splits, and per-task step limits [32]. We keep the family definitions fixed rather than regrouping tasks manually. We train nine task-family modules: find_X, lifespan, conductivity, measurement, growth, state_change, power, chemistry, and life_stages. The first five families have more expert data, with 480, 254, 256, 348, and 145 training variants respectively. Their SFT policies are already near-saturated, so we do not apply RL refinement to them. The remaining four families are lower-data: state_change has 56 training variants, power has 16, chemistry has 52, and life_stages has 10. We apply RL refinement to these four families in the main SciWorld setting. Evaluation is performed over the full ETO dev/test sets used in our experiments, containing 194 seen and 211 unseen variants across the nine families.

7.2 Reward Specification

Table 8 summarizes the immediate per-step rewards used during RL. To make reward construction scalable across task families, we draft the rule sets offline using OpenAI GPT-5.5, prompted with the task description and state-block schema. A representative ALFWorld prompt skeleton is shown in Appendix 7.5. We then implement the proposed rules as fixed deterministic scoring rules. During RL rollouts, no LLM judge or external model is queried. Discounted returns are computed from these rewards with discount factor $\gamma = 0.98$.

7.3 Data Efficiency of Expert Trajectories

To study how many expert trajectories are needed to learn a recurring workflow, we run a data-efficiency sweep on WebShop with Qwen3-8B. We focus on the SFT stage only, because adding RL would introduce additional interaction budget and make the effect of demonstration count harder to isolate. For each setting, we train the same WebShop skill module with the same hyperparameters as the main experiment, varying only the number of expert trajectories used for SFT. Evaluation is performed on a fixed 100-goal test subset, so these numbers are intended to show trends rather than replace the full 500-goal results in Table 1.

Figure 4 shows a clear scaling trend. With only 5 trajectories, the model fails to learn a usable shopping

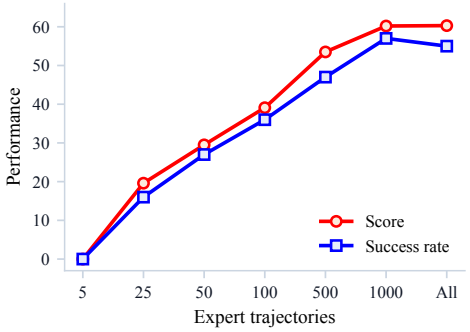

Figure 4: WebShop SFT performance versus expert trajectory count.

Table 7: Training and evaluation data scale, hyperparameters, and evaluation protocol for the main Qwen3-8B experiments. SFT samples denote step-level state-action pairs.

Hyperparameter	ALFWorld	WebShop	SciWorld
Backbone and adapter			
Base model		Qwen3-8B (frozen)	
Adapter		LoRA, rank 64, $\alpha = 128$	
Target modules		q, k, v, o, gate, up, down	
Trainable params		$\sim$175M (2.09% of base)	
Precision		bf16 mixed	
Optimizer		AdamW	
Training data			
Task-family modules	5	1	9
Expert trajectories	2,628	1,571	1,617
SFT samples	36,702	15,087	37,403
Supervised fine-tuning			
Learning rate		2×10^{-4}	
Effective batch		16	
Epochs	3	3	1
Loss masking		action tokens only	
LR scheduler		cosine, warmup ratio 0.1	
LoRA dropout		0.05	
Max seq length	1024	2048	1024
Reinforcement learning			
Initialization		SFT adapter	
Reference model		frozen SFT adapter	
RL epochs		1	
Rollouts per group K		4	
Rollout temperature		0.8	
Rollout top-p		0.95	
Learning rate		5×10^{-6}	
Discount γ		0.98	
KL coefficient β		0.02	
Gradient clip		1.0	
Train games	200/family	200	low-data families only
Max episode steps	30	15	ETO per-task limits
Evaluation			
Eval games (seen/unseen)	140 / 134	500	194 / 211
Performance eval		temperature 0.4, top-p 0.95; seeds 0, 1, 2	
Token-cost eval		greedy ($T = 0$), seed 42	
Max new tokens		64	
Compute			
Hardware		1$\times$A100 80GB	
Training seed		42	
SFT wall-clock	$\sim$173 min	$\sim$143 min	$\sim$90 min
RL wall-clock	$\sim$650 min	$\sim$86 min	$\sim$130 min

policy. Performance improves steadily from 25 to 1,000 trajectories, with success rate increasing from 16.0% to 57.0%. Using all available WebShop trajectories gives similar performance to the 1,000-trajectory setting, suggesting that SFT begins to saturate at this scale for the fixed 100-goal subset.

7.4 Qualitative Examples

Figures 5 and 6 illustrate the same bounded-context interface on WebShop and SciWorld. Each trace shows how the rendered state block, current observation, and one-step context support direct executable action generation without carrying the full episode history.

Table 8: Reward specification used for RL. Rewards are deterministic rules computed from the environment response, action, and tracker state.

Benchmark	Component	Reward terms
ALFWorld	Environment success	$+3.0$ if the episode terminates successfully.
	Progress rewards	$+1.0$ for advancing the tracked subgoal; $+0.02$ for visiting a new hinted location type during search; $+0.05$ for opening a new hinted container; $+0.2$ for reaching the destination; $+0.2$ for opening the destination receptacle; $+0.5$ for a correct placement action.
	Error penalties	-0.3 for invalid actions; -0.2 for no-effect actions when admissible actions are unavailable; -0.2 for repeated no-progress; -0.3 for late-stage regression; -0.2 for looking during placement; -0.3 for moving to a wrong destination instance; -0.1 for revisiting a location; -0.1 for reopening a searched container; -0.15 for wandering after the search phase.
	Step cost	-0.01 per step.
WebShop	Environment success	$+3.0$ for exact successful purchase, i.e., final score $= 1.0$.
	Progress rewards	$+0.15$ for selecting a required option; $+0.10$ when all required options are filled and the phase advances to purchase.
	Error penalties	-0.10 for selecting a wrong option; -0.25 for premature purchase before required options are filled; -0.10 for an $A \to B \to A$ action loop; -0.08 for clicking an already visited product; -0.08 for revisiting the same detail page.
	Step cost	-0.01 per step.
SciWorld	Environment score	$\Delta\text{score}/33.0$, so a 100-point task contributes about $+3.0$ in total; additionally $+1.0$ when the episode terminates with score ≥ 100.
	Progress rewards	Family-specific milestone bonuses tied to the state block, such as focusing or acquiring the target object, reaching the answer room, connecting circuit components, planting and watering seeds, observing state changes, reading recipes, mixing required materials, or focusing the next life stage. Milestone bonuses are one-time and typically range from $+0.08$ to $+2.0$ depending on their position in the procedure.
	Error penalties	Invalid or no-effect actions, repeated actions, oscillations, premature answers, wrong-focus actions, wrong-skill actions, invalid circuit or mixing operations, and task-specific loops. Penalties are typically between -0.03 and -0.25.
	Step cost	-0.01 per step.

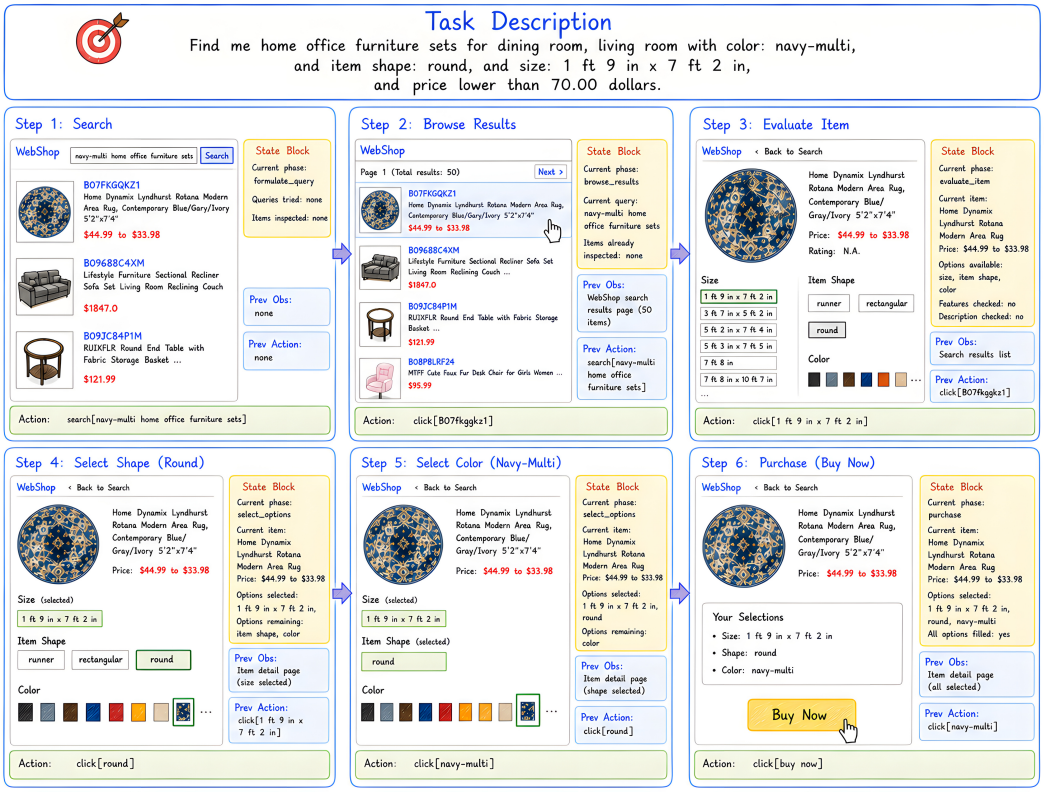

Figure 5: Qualitative WebShop shopping trace with rendered state blocks.

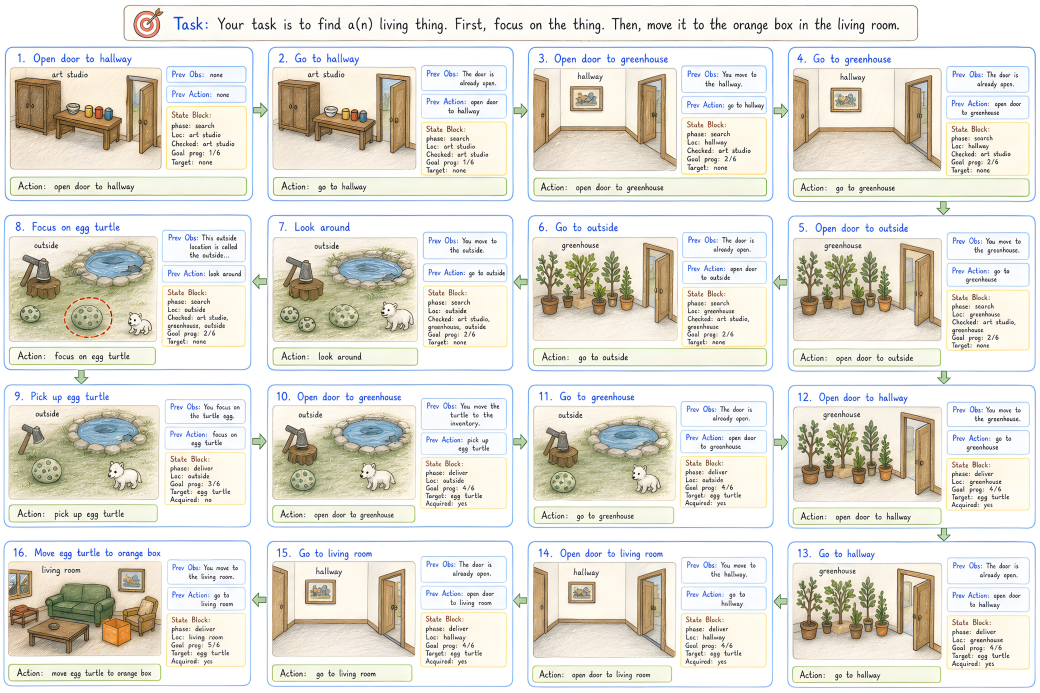

Figure 6: Qualitative SciWorld lab-procedure trace with rendered state blocks.

7.5 Reward-Drafting Prompt Example

To make reward construction reproducible, we release representative reward-drafting prompts. The prompt below shows the ALFWorld pick-and-place template. WebShop and SciWorld use the same structure, with the environment description, action space, and tracker schema replaced by the corresponding task family. The drafted rules are manually checked and then implemented as fixed deterministic scoring rules.

Listing 1: Reward-drafting prompt skeleton (ALFWorld example).

```
SYSTEM
You are drafting deterministic reward-shaping rules for online RL
refinement of an LLM agent. Your job is to design:
(1) r_progress: subgoal-progress rewards aligned with the state block,
(2) r_error: penalties for actions that block, undo, or violate progress,
(3) r_step: a small step cost to discourage unnecessary actions.

All rules must be computable from the tracker state, executed action,
new observation, and updated tracker state. No LLM judge is allowed at
rollout time.

TASK FAMILY
ALFWorld pick-and-place: put a target object into a target receptacle.

ACTION SPACE
go to X; take X from Y; put X in/on Y; open X; close X; examine X; look.

STATE-BLOCK SCHEMA
target_object       : object to manipulate
destination         : target receptacle
holding             : currently held object or None
checked_locations   : visited receptacles
searched_containers : opened containers
current_subgoal     : find_object | take_object | reach_dest |
                      place_object | done
```

```
TRACKER TRANSITION EXAMPLES
- find_object -> take_object when the target object becomes visible.
- take_object -> reach_dest when holding changes from None to target_object.
- reach_dest -> place_object when the agent reaches destination.
- place_object -> done when the object is placed in the destination.
- No progress if the same location is revisited without new evidence.
- Regression if the agent leaves the destination during place_object.

DESIGN INSTRUCTION
Design reward-shaping terms over the above state-block fields and
tracker transitions. The total reward will be:

    r_t = r_env + r_progress - r_error - r_step

where r_env is fixed by the environment. Return a Python-style
dictionary reward_terms.

Rules:
- r_progress should reward tracker-state changes that advance the
  current subgoal or make required state-block fields closer to done.
- r_error should penalize invalid actions, repeated no-progress actions,
  premature placement, wrong-destination behavior, and subgoal regression.
- r_step should be a small positive penalty magnitude applied each step
  to discourage unnecessary wandering or browsing.
- Every trigger must explicitly reference tracker fields, actions,
  observations, or tracker-state changes.
```

7.6 Generalization Probe Protocol

We evaluate the Qwen3-8B SFT+RL adapters used in Table 1, without additional training or checkpoint selection. WebShop uses environment seed 2, all 500 standard test goals unless filtered below, and inference seeds 0, 1, and 2. ALFWorld uses all 134 official unseen games with the same three inference seeds. Decoding follows the main performance protocol: temperature 0.4 and top-p 0.95. Each probe is also evaluated with the original frozen Qwen3-8B model, with the same tracker, prompt, parser, task instances, and decoding settings.

Instruction paraphrasing. GPT-5.4 rewrites each instruction under a fixed JSON output schema, preserving constraint tokens such as attribute values, sizes, quantities, and price inequalities. The response includes an explicit constraint-preservation check, and a random sample is manually checked. Only the instruction shown to the policy is replaced: the environment goal, reward computation, and tracker parsing use the original instruction. The probe therefore tests the policy's response to wording changes, rather than robustness of the tracker parser to paraphrases. The construction is applied to all 500 WebShop test instructions and all 134 ALFWorld unseen instructions.

Unseen constraint combinations. We extract constraint sets from the 1,571 WebShop SFT training goals and retain test goals whose complete joint constraint set does not occur in the SFT goals, although each atomic constraint does. This yields 446 test goals. No LLM is used for this filtering. Because this is a subset of the original 500-goal evaluation, its absolute success rate should not be interpreted as a paired difference from the full-set result. The probe tests combinations absent from the SFT goals, without establishing absence from the backbone's pretraining data or subsequent RL experience.

State-block reserialization. We retain every WebShop tracker field and value but render the block as sorted JSON, changing its ordering and delimiters. The tracker update rules, environment, and policy weights are unchanged. This isolates sensitivity to the serialization of the supplied state.

Table 9 supplements the success rates in Table 4 with WebShop scores. The frozen base is weak under both the standard and perturbed interfaces. This isolates a substantial contribution from adaptation, while the remaining paraphrase drop shows that the learned policy is still sensitive to some changes in wording. None of these probes changes the environment dynamics or establishes transfer to a new task family.

Table 9: WebShop scores for the generalization probes (mean±std over three inference seeds). Success rates and sample sizes appear in Table 4.

Input condition	Frozen-base score	SFT+RL score
Original	27.3 ± 0.9	84.0 ± 0.0
Paraphrased instruction	28.2 ± 0.8	82.7 ± 0.5
Unseen constraint combination	27.3 ± 0.8	84.9 ± 0.3
State block as sorted JSON	17.4 ± 0.6	83.3 ± 0.4

7.7 Frozen-Base Controls

To isolate the contribution of the learned adapter under the supplied state interface, we evaluate the original frozen Qwen3-8B model without loading any LoRA. Task instances, splits, tracker, state block, prompt template, action parser, environment, decoding parameters, and inference seeds are otherwise unchanged. Table 10 compares the standard evaluations with the original SFT and SFT+RL results. All values are mean±std over the three inference seeds of the main performance protocol.

Table 10: Qwen3-8B with the same state interface before and after adapter training. Entries give Score / SR (%); ALFWorld reports SR only.

Benchmark	Split	Frozen base	SFT	SFT+RL
ALFWorld	Seen	0.0 ± 0.0	62.4 ± 2.5	83.6 ± 1.8
	Unseen	0.0 ± 0.0	61.7 ± 2.2	89.6 ± 0.4
WebShop	Test	27.3 ± 0.9 / 22.5 ± 0.8	69.5 ± 1.7 / 62.2 ± 1.2	84.0 ± 0.0 / 76.8 ± 0.2
SciWorld	Seen	11.6 ± 1.2 / 4.8 ± 0.3	65.7 ± 1.6 / 58.2 ± 1.5	72.8 ± 2.7 / 62.9 ± 2.2
	Unseen	12.3 ± 0.3 / 4.9 ± 0.7	61.5 ± 0.9 / 52.0 ± 0.7	79.7 ± 0.2 / 66.4 ± 0.5

Table 11 extends this control to the generalization probes. WebShop score counterparts are reported in Table 9. The base model reaches only 13.5–22.6% SR under the WebShop perturbations and 0.0% on ALFWorld, despite receiving the same tracker and state information. The adapted policy's performance and robustness therefore exceed those of the unadapted backbone. These controls concern the tested task families and do not establish transfer to new environments.

Table 11: Frozen-base controls for the generalization probes with Qwen3-8B (mean±std over three inference seeds). The frozen base and trained policy receive the same state interface. N is the number of goals per seed; SR is in percent.

Benchmark	Input condition	N	Frozen-base SR	SFT+RL SR
WebShop	Original	500	22.5 ± 0.8	76.8 ± 0.2
	Paraphrased instruction	500	22.5 ± 0.6	72.2 ± 0.7
	Unseen constraint combination	446	22.6 ± 0.9	75.8 ± 0.6
	State block as sorted JSON	500	13.5 ± 0.2	75.7 ± 0.8
ALFWorld	Original unseen	134	0.0 ± 0.0	89.6 ± 0.4
	Paraphrased unseen	134	0.0 ± 0.0	89.1 ± 1.1

7.8 LLM-Generated WebShop Tracker

We test whether the state interface can be constructed with an LLM by replacing the authors' WebShop tracker with an LLM-generated implementation. GPT-5.4 receives the public text-mode environment contract, aggregate coverage and selected examples from the 1,571 public SFT trajectories, and the public option-group vocabulary. The authors' manual tracker and evaluation goals 0–499 are excluded from generation. The model produces a state schema, deterministic parser and update logic, bounded JSON renderer, prompt builder, action extractor, and dense reward rules. Integration repairs are also LLM-produced, with deterministic trajectory replay supplying validation signals.

Validation replays all 15,111 states from the training trajectories, including 6,523 item-page states. It checks markup removal, product-title extraction, separation of titles from selectable options, and tracker execution. The finalized implementation records zero failures on these checks and makes

no LLM, API, or network call during training or inference. These checks cover the observed replay states; they do not guarantee correct tracking on every possible future state.

We train a Qwen3-8B LoRA for three SFT epochs, then refine it on 200 training games with four rollouts per game. The run uses one A100 80GB and training seed 42. Both checkpoints are evaluated on all 500 held-out goals with greedy decoding, seed 42, and a 15-step limit. Table 12 reports this single-run proof of concept alongside the original manual-tracker results. The two implementations use the same source expert corpus, with tracker-specific preprocessing; the manual-tracker reference is a three-inference-seed mean, so this is a descriptive comparison.

Table 12: WebShop performance with manual and LLM-generated trackers. Manual-tracker numbers are the original three-seed means; generated-tracker numbers are single greedy evaluations on 500 goals.

Tracker and training stage	Score	SR (%)
Manual tracker, SFT	69.5	62.2
Manual tracker, SFT+RL	84.0	76.8
LLM-generated tracker, SFT	72.6	61.2
LLM-generated tracker, SFT+RL	75.0	65.8

RL improves success by 4.6 percentage points with the generated tracker. The final 65.8% SR shows that an LLM-generated interface can support the complete training pipeline, but remains 11.0 points below the mature tracker reference. This is evidence for feasible offline construction followed by validation, rather than parity with manual engineering or autonomous discovery of useful state representations in arbitrary domains.

Generation request. The generation prompt supplies the environment's page types, action syntax, option-group vocabulary, trajectory-coverage summary, and representative transitions. It requests Python tracker code and tests, the state schema, deterministic reward rules, assumptions, and a review checklist under a fixed output schema. The core requirements are summarized below. The environment-specific examples are supplied separately.

Listing 2: Condensed requirements supplied for offline tracker generation.

```
INPUTS
Public WebShop text-interface contract, option-group vocabulary,
training-trajectory coverage, and representative transitions.
Do not use the authors' tracker or held-out evaluation goals.

OUTPUTS
State schema; deterministic Python parser and update logic;
bounded JSON renderer; prompt builder and action extractor;
dense reward rules; tests; assumptions and review checklist.

CONSTRAINTS
Use observable actions, pages, and tracked progress.
No model, API, or network calls in the generated runtime.
Compute dense rewards from observable transitions and terminal score.
Validate against training-trajectory replay and repair failures.
```

7.9 A Unified LoRA Across Three Benchmarks

To examine whether per-family modules are an architectural requirement, we train one Qwen3-8B LoRA jointly on SFT examples from ALFWorld, WebShop, and SciWorld, using benchmark-balanced sampling. The same adapter is evaluated through each benchmark's native environment and tracker. This experiment changes parameter sharing while retaining the benchmark-specific state interfaces; it does not test execution in an unseen environment. The adapter is SFT-only, with no RL refinement, and the results are single evaluation runs. The SciWorld run uses alias-only action compilation with its action guard disabled.

Table 13 compares this adapter with the independent Qwen3-8B SFT modules from Table 1. WebShop remains close to the independent-module reference and SciWorld improves on both splits. ALFWorld

unseen success falls by 6.5 points, indicating interference despite benchmark-balanced sampling. The unified numbers are single runs and the independent-module numbers are inference-seed means, so these differences are descriptive, rather than significance tests.

Table 13: Independent and cross-benchmark unified Qwen3-8B SFT adapters. Entries give Score / SR (%); ALFWorld reports SR only. Independent values are mean±std; unified values are single-run results. Neither column includes RL refinement.

Benchmark	Split	Independent SFT	Unified SFT
ALFWorld	Seen	62.4 ± 2.5	65.7
	Unseen	61.7 ± 2.2	55.2
WebShop	Test	69.5 ± 1.7 / 62.2 ± 1.2	70.2 / 60.8
SciWorld	Seen/dev	65.7 ± 1.6 / 58.2 ± 1.5	70.2 / 64.4
	Unseen/test	61.5 ± 0.9 / 52.0 ± 0.7	71.6 / 56.4

Together with the within-ALFWorld unified module in Section 4.2, this result shows that one adapter can support multiple task families. Per-family deployment retains a practical benefit: adding a workflow updates only its new adapter, leaving existing deployed adapters fixed. The unified result demonstrates joint learning across heterogeneous interfaces, while the remaining ALFWorld interference motivates retaining specialization as a deployment option.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the formulation, training pipeline, evaluation scope, and main empirical claims. The reported claims are supported by the experiments in Section 4 and the limitations are discussed in Section 6.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 discusses limitations in tracker/reward automation, text-only evaluation, and transfer to new skills.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present theoretical theorems or formal proofs. The equations define the problem setup, model input, losses, and reward/optimization objectives.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The method, algorithms, training stages, reward construction, data splits, hyperparameters, and evaluation settings are described in Sections 3–4 and Appendix 7.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide code and reproduction instructions at `https://github.com/haoyangxie/constant-context-skill-learning`. The repository includes training and evaluation scripts, tracker/reward implementations, and configuration files for the main experiments.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Section 4.1 describes the benchmarks, baselines, model backbones, and evaluation protocol. Appendix 7.1 reports the full training data scale, hyperparameters, decoding settings, and compute setup.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: The main performance results are reported as mean$\pm$standard deviation over three inference seeds, as stated in Section 4.1 and Table 1. Token-cost measurements use deterministic decoding for stable accounting.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix 7.1 reports the GPU type, precision, training seeds, and wall-clock time for the main SFT and RL runs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses public benchmark environments and open model backbones, does not involve human subjects, and does not collect private user data. We have reviewed and followed the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The motivation and conclusion discuss privacy, cost, and deployment implications for personal agents. Section 6 also notes limitations and future directions for safer and more general deployment.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk pretrained models, scraped datasets, or generative media assets. Experiments use public text-based benchmark environments and trained task adapters for those benchmarks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the benchmark environments and model backbones used in the experiments. We use these existing assets under their respective licenses and terms.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We will release code and trained task-family adapters with documentation, reproduction scripts, and configuration files.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The experiments do not involve crowdsourcing or human-subject studies. Expert trajectories are obtained from benchmark data or prior successful benchmark executions.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The work does not involve human subjects, so IRB approval or equivalent review is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are central to the method: the base policies are LLM backbones, and Appendix 7.2 states that a strong reasoning model is used offline to draft deterministic reward rules. No LLM judge is queried during RL rollouts.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.