vSFT: Learning to Verbally Tune Agents at Test Time

Anonymous authors
Paper under double-blind review

Abstract

Large language model agents are increasingly improved via agentic reinforcement learning, which can be on-policy but suffers from expensive rollouts and sparse rewards, limiting its application at test time. In this paper, we present a **v**erbally **S**upervised **F**ine-**T**uning (vSFT) framework that enables the conjunction of dense feedback in natural language and weight updates at test time. Specifically, our vSFT forms a closed loop between an agent and a supervisor via a hypernetwork, in which the supervisor generates verbal supervision from the agent's failed trajectories, and the hypernetwork converts that supervision into low-rank weight updates to tune the agent. We validate that this test-time in-parameter evolving process outperforms both its in-context counterparts and prevalent reinforcement learning methods. In particular, our vSFT using Qwen3-4B outperforms GRPO and Reflexion by 11.5% and 10.8% in success rate on ALFWorld's in-distribution split, and by 4.8% and 18.2% on WebShop, respectively.

1 Introduction

Modern autonomous agents use large language models (LLMs) as policies for interactive decision-making, enabling them to plan, act, call tools, and perceive the environment over multiple turns (Yao et al., 2023). Although pretrained LLMs possess strong general-purpose capabilities, they are not natively optimized for specific agent tasks, requiring to be further adapted via post-training methods for parameters updating (Jiang et al., 2025).

Recently, supervised fine-tuning (SFT) and reinforcement learning (RL) have been widely used for post-training LLM agents (Zhang et al., 2026a). In particular, on-policy distillation (OPD) becomes a promising paradigm (Wang et al., 2026; Lu et al., 2026). Given golden trajectories, SFT is simple and data-efficient for forcing token-level imitation across positions in parallel, however, it is vulnerable to distribution shift on unseen trajectories due to the inherent off-policy nature (Nie, 2026). Moreover, it could also be problematic to collect golden trajectories for open-world agent tasks. In contrast, RL is on-policy, evolving the agent from its own rollouts rather than imitating from a fixed trajectory dataset. Nonetheless, RL often relies on sparse rewards at trajectory level and GRPO-style methods (Shao et al., 2024; Feng et al., 2025; Chu et al., 2026) which further increase the rollout costs due to multiple sampled trajectories.

As a lightweight alternative to RL fine-tuning, Shinn et al. (2023) introduced a verbal reinforcement learning (vRL), which reinforces agents at test time through in-context reflections. Unlike post-training methods that directly update weights and thereby improve the agents essentially, however, vRL relies on in-context learning, which incurs recurring context overhead and may become less effective in long-horizon tasks. Similar to prior work (Zhao et al., 2025; Geng et al., 2026), we observe that, regardless of whether the reflections could help complete the task, the agent often struggles to follow them, suggesting that in-context reflections cannot reliably steer the agent's behavior (as shown in Fig. 1). This motivates us to develop an end-to-end trainable method that learns to internalize verbal feedback into model parameters to reduce context overhead and enable more reliable behavioral adaptation.

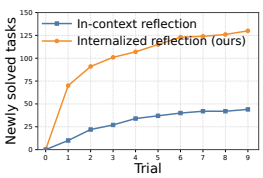

Figure 1: **Efficacy Comparison.** On the shared task set that both methods fail in the initial trial, we replace the in-context reflections with our internalized reflections without altering their content to fairly compare the effectiveness in guiding the agent in a controlled setting.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: A summary of existing methods for improving LLM agents with respect to four desirable properties. Our vSFT is the only method that satisfies all of them. As discussed in §3.1, vSFT can be viewed as an extension of OPD that requires neither access to next-token distributions nor backpropagation once the hypernetwork has been trained, thereby enabling test-time adaptation under supervision from closed-source API models.

Method	On policy	In parameter	Dense feedback	W/o backprop.
SFT	✗	✓	✓	✗
RL	✓	✓	✗	✗
OPD	✓	✓	✓	✗
vRL (Shinn et al., 2023)	✓	✗	✓	✓
JitRL (Li et al., 2026b)	✓	✗	✗	✓
vSFT	✓	✓	✓	✓

Inspired by recent works in hypernetwork-generated LoRA adapters that convert natural language into low-rank weight updates (Charakorn et al., 2025; 2026), in this paper, we propose **vSFT**, a **v**erbally **S**upervised **F**ine-**T**uning framework for tuning agents at test time via verbal supervision, as illustrated in Fig. 2. To be specific, in vSFT, we build an iterative closed loop among agent, environment and supervisor via a trained lightweight hypernetwork, in which the agent interacts with the environment, the supervisor model provides verbal supervision (i.e., verbal feedback, such as skills and reflections), and the hypernetwork converts the verbal supervision into weight updates in the form of a LoRA adapter to tune the agent. Here, the hypernetwork serves as a parameterized, learnable interface through which the agent iteratively improves itself in response to self-reflection or external feedback, supplying a mechanism pretrained LLMs otherwise lack (Zweiger et al., 2026).

As summarized in Tab. 1, our vSFT is the only method that satisfies all of the four desired properties: a) on-policy (i.e., at test-time) improvement from the agent's own rollouts, reducing the train-test mismatch; b) in-parameter adaptation rather than in-context learning, enabling more reliable and context-efficient steering[1]; c) using dense feedback, mitigating the credit assignment problem in agentic RL (Zhang, 2026); and d) backpropagation-free test-time fine-tuning, rather than being restricted to offline post-training.

In experiments, we find that our vSFT outperforms both its in-context counterparts and prevalent reinforcement learning methods. In particular, our vSFT using Qwen3-4B outperforms GRPO and Reflexion by 11.5% and 10.8% in success rate on ALFWorld's in-distribution split, and by 4.8% and 18.2% on WebShop, respectively, while using only 4% of GRPO's training rollout budget on ALFWorld and 10% on WebShop. Moreover, when further using Qwen-Plus as the supervisor, our vSFT substantially outperforms Qwen-Plus and performs on par with Qwen3.8-Max, despite the agent having under 5% as many active parameters.

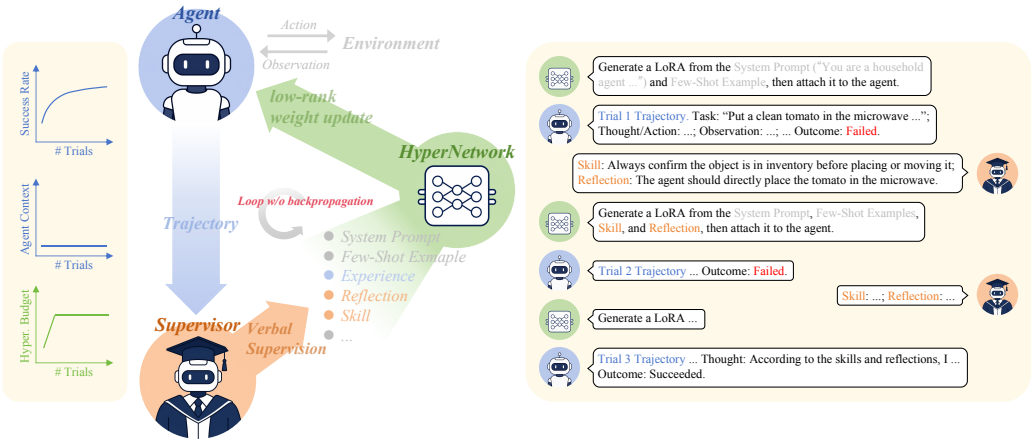

Figure 2: **Overview of our proposed vSFT.** Given a task, the hypernetwork first internalizes the system prompt and few-shot examples into an initial LoRA adapter, and then incorporates the supervisor's feedback through subsequent trial and error.

[1]Human intelligence and self-improvement are not driven by an ever-growing context but through the changes in neural connections (e.g., Hebbian learning), much as weight updates in neural networks.

Paper Contribution. The main contributions of the paper are highlighted as follows.

1. We propose vSFT, a novel framework that iteratively improves an agent at test time through a closed loop of the agent's behavior, the supervisor's verbal feedback, and the weight updates to the agent, thereby enabling a test-time parameter adaptation (§3.1).
2. We provide an effective and parameterized mechanism for agent self-improvement by using a trained hypernetwork to internalize verbal feedback into the agent's parameters without back-propagation (§3.2).
3. We empirically validate the efficacy of our vSFT and exhibit its strong generalization ability and robustness (§4).

2 Related Work

2.1 LLM Agents and Their Adaptation

Large language models (LLMs) have enabled the development of autonomous agents by serving as policies for interactive decision-making. Nonetheless, even capable pretrained LLMs often require further adaptation to specific agentic tasks. While such adaptation can target either the agent itself or other callable components, in this paper, we focus on adapting the agent itself.

Existing adaptation methods are mainly based on post-training (i.e., fine-tuning). Supervised fine-tuning (SFT) suffers from limited generalization due to its off-policy nature (Chen et al., 2026; Shenfeld et al., 2026), whereas reinforcement learning (RL) can be deployed on-policy but demands expensive rollouts. On the other hand, lightweight test-time alternatives have recently attracted increasing attention. For example, verbal reinforcement learning (vRL) (Shinn et al., 2023; Kim et al., 2025) adapts agents through verbal reflections and in-context learning rather than scalar rewards and in-parameter updates; just-in-time reinforcement learning (JitRL) (Li et al., 2026b) estimates action advantages from experiences. Unlike these in-context or action-space alternatives, our vSFT bridges verbal feedback and parameter adaptation through a hypernetwork. Similar to on-policy distillation (OPD) (Agarwal et al., 2024), vSFT can be viewed as an on-policy extension of SFT. However, OPD is typically applied during post-training, whereas vSFT runs its closed loop at test time and can therefore complement post-training methods.

2.2 Hypernetwork-generated LoRAs

Hypernetwork-generated LoRAs (HyperLoRAs) has recently emerged as a promising approach for instant task adaption (Charakorn et al., 2025) and knowledge injection (Charakorn et al., 2026; Liu et al., 2026), in which textual context (e.g., task descriptions or documents) is converted into LoRA adapters by a specially trained hypernetwork. Such a context-to-parameter paradigm offers two key advantages: a) adapters can be represented, stored, and edited in natural language, while being instantiated as LoRAs only when needed; and b) new adapters can be generated for unseen tasks (or documents) instantly without additional training. Soon then, follow-up work extends HyperLoRAs to repository internalization for code language models (Hotsko et al., 2026) and to skill internalization for agents (Yu et al., 2026; Zhang & Qi, 2026). Our vSFT instead explores a new application for HyperLoRA, i.e., using it as a learnable component that enables an agent to iteratively update its weights at test time in response to verbal feedback from a supervisor.

3 Methods

In this section, we present the proposed vSFT framework, which consists of two separate parts: a) *test-time in-parameter agent adaptation* (§3.1), which is a closed-loop process in which the hypernetwork iteratively adapts the agent by converting verbal feedback from a supervisor, who diagnoses the agent's failures, into weight updates to the agent; and (b) *offline hypernetwork meta-training* (§3.2), in which the hypernetwork is trained to generate a LoRA adapter that effectively steers the agent's behavior. In addition, we also discuss the connections between vSFT and OPD and the potentials of the hypernetwork to serve as a back-propagation-free optimizer.

Algorithm 1 verbally Supervised Fine-Tuning (vSFT)

Models: base agent $\boldsymbol{\pi}_{\boldsymbol{\theta}}$, supervisor $\mathcal{S}$, hypernetwork $\boldsymbol{h}_{\boldsymbol{\phi}}$
Input: system prompt $\boldsymbol{p}$, few-shot examples $\mathcal{F}$, current task instruction $\boldsymbol{x}$, reference task instruction $\tilde{\boldsymbol{x}}$ and its trajectory $\tilde{\boldsymbol{\tau}}$
Hyperparameters: FIFO size B, maximum number of trials T

Initialize the accumulated verbal feedback (i.e., memory): $\mathcal{M}^{(0)} \leftarrow \emptyset$.
for $t = 0$ to $T - 1$ **do**
  Compose the condition (i.e., hypernetwork input) for this trial: $\boldsymbol{c}^{(t)} \leftarrow (\boldsymbol{p}, \mathcal{F}, \mathcal{M}^{(t)})$.
  Internalize the condition into the agent's parameters: $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta} \oplus \boldsymbol{h}_{\boldsymbol{\phi}}(\boldsymbol{c}^{(t)})$.
  Roll out $\boldsymbol{\pi}_{\boldsymbol{\theta}^{(t)}}$ on $\boldsymbol{x}$ to obtain trajectory $\boldsymbol{\tau}^{(t)}$ and outcome $y^{(t)} \in \{\text{SUCCESS}, \text{FAILURE}\}$.
  if $y^{(t)} = \text{SUCCESS}$ **then**
    return $\boldsymbol{\tau}^{(t)}$
  else
    Diagnose the failure and generate verbal feedback: $\boldsymbol{z}^{(t)} \leftarrow \mathcal{S}(\boldsymbol{p}, \boldsymbol{x}, \boldsymbol{\tau}^{(t)}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{\tau}}, \mathcal{M}^{(t)})$.
    Update memory with a FIFO queue of size B: $\mathcal{M}^{(t+1)} \leftarrow \text{FIFO}_B(\mathcal{M}^{(t)}, \boldsymbol{z}^{(t)})$.
  end if
end for
return $\boldsymbol{\tau}^{(T-1)}$

3.1 Test-time in-parameter agent adaptation

3.1.1 Preliminaries and Notation

We denote an agent as $\boldsymbol{\pi}_{\boldsymbol{\theta}}$, with $\boldsymbol{\theta}$ denoting its pretrained parameters. The agent attempts to solve a task specified by a task instruction $\boldsymbol{x}$ (e.g., "*put a clean tomato in the microwave*" in a household setting), given a system prompt $\boldsymbol{p}$ and few-shot examples $\mathcal{F}$. The agent is allowed up to T trials until success. After a failed trial, a supervisor $\mathcal{S}$ diagnoses the trajectory and provides verbal feedback to guide the agent in the next trial. Unlike vRL (Shinn et al., 2023), where verbal feedback is injected into the agent's context while its parameters remain unchanged across trials, our vSFT employs a hypernetwork $\boldsymbol{h}_{\boldsymbol{\phi}}$ to internalize verbal feedback into the agent's parameters, updating them across trials. We therefore denote the agent at trial t as $\boldsymbol{\pi}_{\boldsymbol{\theta}^{(t)}}$, where $\boldsymbol{\theta}^{(t)}$ denotes its updated parameters.

3.1.2 Closed loop among agent, supervisor, and hypernetwork

Specifically, our vSFT constructs a closed loop at test time among the agent, the supervisor, and the hypernetwork, as illustrated in Fig. 2. Next, we will introduce the three components of our vSFT and describe how they interact with each other within the loop.

Agent. The agent at trial t interacts with the environment over multiple steps, yielding a trajectory

$$\boldsymbol{\tau}^{(t)} = (\boldsymbol{o}_1^{(t)}, \boldsymbol{a}_1^{(t)}, \boldsymbol{o}_2^{(t)}, \boldsymbol{a}_2^{(t)}, \ldots, \boldsymbol{o}_{s_t}^{(t)}, \boldsymbol{a}_{s_t}^{(t)}, y^{(t)}), \tag{1}$$

where $\boldsymbol{o}_s^{(t)}$ and $\boldsymbol{a}_s^{(t)}$ denote the observation and the action at the s-th interaction step, respectively, and $y^{(t)}$ denotes the final outcome (e.g., success or failure). An action is sampled from the current policy $\boldsymbol{\pi}_{\boldsymbol{\theta}^{(t)}}$ as

$$\boldsymbol{a}_s^{(t)} \sim \boldsymbol{\pi}_{\boldsymbol{\theta}^{(t)}}\left(\cdot \mid \boldsymbol{x}, \boldsymbol{o}_{\leq s}^{(t)}, \boldsymbol{a}_{<s}^{(t)}\right), \tag{2}$$

where agent $\boldsymbol{\pi}_{\boldsymbol{\theta}^{(t)}}$ observes only the task instruction and the within-trial interaction history up to s. That is, in vSFT, the system prompt, few-shot examples, skills, reflections, and other forms of memory are internalized into $\boldsymbol{\theta}^{(t)}$, rather than being injected into the agent's context across trials.

Supervisor. The supervisor $\mathcal{S}$ can be either the agent itself or a more capable model. If trial t fails, the supervisor diagnoses the failed trajectory and generates the verbal feedback $\boldsymbol{z}^{(t)}$ as

$$\boldsymbol{z}^{(t)} = \mathcal{S}(\boldsymbol{p}, \boldsymbol{x}, \boldsymbol{\tau}^{(t)}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{\tau}}, \mathcal{M}^{(t)}), \tag{3}$$

where $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{\tau}}$ denote a successfully solved task and its corresponding trajectory, respectively, and $\mathcal{M}^{(t)}$ denotes the accumulated verbal feedback (i.e., memory) from previous trials. The verbal feedback essentially consists of *a skill* and *a reflection*—skill encodes reusable procedural knowledge

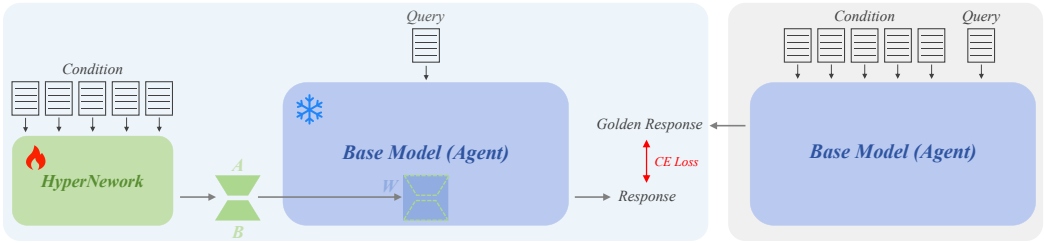

Figure 3: **Hypernetwork meta-training pipeline.** The curated dataset (right panel) is essentially an SFT dataset, but consists of condition-query-response triplets rather than query-response pairs. This design trains the hypernetwork to convert verbal feedback into LoRA adapters that effectively steer the agent's behavior (left panel).

that is transferable across tasks and reflection diagnoses what went wrong in the current trajectory. Following vRL (Shinn et al., 2023), we maintain the accumulated verbal feedback $\mathcal{M}^{(t)}$ in a "*first-in, first-out*" (FIFO) queue, i.e.,

$$\mathcal{M}^{(t+1)} = \mathrm{FIFO}_B(\mathcal{M}^{(t)}, \boldsymbol{z}^{(t)}), \quad \text{with } \mathcal{M}^{(0)} = \emptyset, \tag{4}$$

where FIFO_B appends $\boldsymbol{z}^{(t)}$ to $\mathcal{M}^{(t)}$ while retaining only the most recent B entries.

Hypernetwork. We refer to the hypernetwork input, i.e., the textual content to be internalized into the agent's parameters by the hypernetwork as the *condition*.[2] Specifically, we define the condition for trial t as

$$\boldsymbol{c}^{(t)} = (\boldsymbol{p}, \mathcal{F}, \mathcal{M}^{(t)}). \tag{5}$$

Then, given the condition $\boldsymbol{c}^{(t)}$, the hypernetwork $\boldsymbol{h}_{\boldsymbol{\phi}}$ generates a trial-specific LoRA adapter, i.e.,

$$\left\{\left(\boldsymbol{A}_k^{(t)}, \boldsymbol{B}_k^{(t)}\right)\right\}_{k\in\Omega} = \boldsymbol{h}_{\boldsymbol{\phi}}(\boldsymbol{c}^{(t)}), \tag{6}$$

where Ω denotes the index set of target weight matrices. For each $k \in \Omega$, let $\boldsymbol{W}_k \in \mathbb{R}^{d_k^{\mathrm{out}} \times d_k^{\mathrm{in}}}$ denote the corresponding weight matrix, $\boldsymbol{A}_k^{(t)} \in \mathbb{R}^{r \times d_k^{\mathrm{in}}}$ and $\boldsymbol{B}_k^{(t)} \in \mathbb{R}^{d_k^{\mathrm{out}} \times r}$ denote the generated low-rank factor matrices. We then induce the effective weight update for trial t as follows:

$$\boldsymbol{W}_k^{(t)} = \boldsymbol{W}_k + \alpha \boldsymbol{B}_k^{(t)} \boldsymbol{A}_k^{(t)}, \tag{7}$$

where α is a scaling coefficient. While the agent's pretrained parameters $\boldsymbol{\theta}$ remain unchanged throughout test time, the generated adapter changes across trials. In our implementation, each newly generated adapter replaces the previous one rather than being appended to it. Thus, the effective agent parameters at trial t can be written as

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta} \oplus \boldsymbol{h}_{\boldsymbol{\phi}}(\boldsymbol{c}^{(t)}), \quad \forall t \in \{0, 1, \ldots, T-1\}, \tag{8}$$

where $\oplus$ denotes equipping the generated adapter.

Discussion. Intriguingly, we find that vSFT has a close conceptual connection to OPD. To be specific, both of vSFT and OPD mitigate the off-policy limitation of SFT by adapting the agent on states induced by its own rollouts. Nevertheless, the distinction lies in when their adaptation occurs: OPD is during post-training, whereas vSFT is at test-time. These two stages are complementary and can be applied in conjunction. Moreover, both methods are distilling a teacher's (i.e., supervisor in our context) knowledge or capability into the student's parameters. However, OPD favors consistent thinking patterns between the student and the teacher (e.g., high overlap in their top-k next-token sets) for effective optimization, making it particularly sensitive to teacher–student mode mismatch (Li et al., 2026a). In contrast, vSFT directly internalizes the teacher's verbal feedback, thereby avoiding explicit token-level alignment. In a degree, our vSFT thus suggests a promising test-time extension of OPD, with a more general and interpretable supervision interface.

[2]Since our vSFT is compatible with different base models and their associated hypernetworks (e.g., D2L (Charakorn et al., 2026) and SHINE (Liu et al., 2026)), we abstract away the specific architectural and implementation details of the hypernetwork and characterize it solely by its input and output.

3.2 OFFLINE HYPERNETWORK META-TRAINING

While vSFT runs a closed loop at test time without back-propagation, the hypernetwork that bridges natural language and weight updates requires SFT-style offline training. In another words, the conventional SFT is to *directly adapt a model*, whereas the hypernetwork in our vSFT is to *learn how to adapt the model*. We therefore view this process as *meta-training*, in which the self-improvement[3] mechanism is learned offline and applied at test time. Specifically, during meta-training, the hypernetwork learns to generate LoRA adapters that enable the agent to reproduce behavior previously elicited by in-context conditions (as defined in Eq. 5). These conditions are now provided only to the hypernetwork rather than to the agent, as illustrated in Fig. 3.

Data Curation. We first run the agent $\boldsymbol{\pi_\theta}$ (alternatively another agent) in a vRL (Shinn et al., 2023) framework to interact with the environment and collect its trajectories, where the conditions are directly provided in context (see the right panel of Fig. 3). Then, we construct a dataset of triplets, each consisting of a condition, a query (prefix), and a response, i.e., $(\boldsymbol{c}, (\boldsymbol{x}, \boldsymbol{o}_{\le s}, \boldsymbol{a}_{<s}), \boldsymbol{a}_s)$, extracted from the successful trajectories:

$$\mathcal{D} = \left\{ \left(\boldsymbol{c}^i,\ \boldsymbol{q}_s^i,\ \boldsymbol{a}_s^i \right) \mid \tau^i \in \mathcal{T}_{\text{succ}},\ s = 1, \ldots, s_i \right\}, \tag{9}$$

where i indexes completed tasks rather than trials, and $\boldsymbol{q}_s^i = (\boldsymbol{x}^i, \boldsymbol{o}_{\le s}^i, \boldsymbol{a}_{<s}^i)$. Since we retain only the final trial for each completed task, the trial index is uniquely determined by i and is therefore omitted from Eq. 9.

Loss Function. During meta-training, the condition is removed from the agent's context and is provided only to the hypernetwork. The hypernetwork then generates a LoRA adapter (as in Eqs. 6, 7, and 8), while the adapted agent aims to imitate the actions in the successful trajectory using only the task instruction and interaction history as context, with the condition internalized in its weights:

$$\mathcal{L}(\boldsymbol{\phi}) = -\frac{1}{\sum_{i=1}^{|\mathcal{T}_{\text{succ}}|} s_i} \sum_{i=1}^{|\mathcal{T}_{\text{succ}}|} \sum_{s=1}^{s_i} \log \pi_{\boldsymbol{\theta} \oplus \boldsymbol{h}_{\boldsymbol{\phi}}(\boldsymbol{c}^i)} \left(\boldsymbol{a}_s^i \mid \boldsymbol{q}_s^i \right), \tag{10}$$

where only hypernetwork parameters $\boldsymbol{\phi}$ are trainable.

For clarity, we summarize the procedure of our vSFT in Alg. 1. Note that in vSFT the condition plays a role similar to privileged information in on-policy self-distillation (OPSD) (Penaloza et al., 2026), in which OPSD learns a student policy that internalizes such privileged information through distillation, whereas our vSFT adopts a meta-learning viewpoint that learns how to transform privileged verbal feedback into low-rank weight updates. This allows vSFT to learn adaptation from offline trajectories while performing it at test time. From another perspective, the verbal feedback in vSFT can be viewed as a semantic counterpart of the error signal used in back-propagation. Unlike conventional error signals derived from predefined loss functions, it is produced by an LLM and expressed in natural language. The hypernetwork then learns to convert such verbal signals into weight updates, acting as a back-propagation-free optimizer.

4 EXPERIMENTS

In this section, we evaluate vSFT through the following research questions motivated by our work.

1. Can vSFT outperform in-context learning counterparts and reinforcement learning methods while yielding more effective self-improvement than in-context reflection?
2. Can vSFT enable a weaker agent to match or surpass the performance of a stronger supervisor while remaining robust to supervisor choice across model families?
3. Can meta-training make the hypernetwork more resistant to overfitting under extended training and enable persistently internalized verbal feedback that generalizes to new scenarios?

We next introduce the experimental setup and implementation details, and then present and analyze our findings, highlighting how they collectively address these questions.

[3] We define self-improvement as the ability of a system (e.g., the agent and hypernetwork together in our framework) to improve its own performance (i.e., "doing") in response to self-reflection or external feedback, rather than merely generate high-quality reflections (i.e., "knowing").

Table 2: **Comparisons on ALFWorld.** We compare vSFT against ICL and RL methods across different base models on the seen and unseen validation splits of ALFWorld, reporting success rates for six categories of tasks and the overall and averaged results. The top three distinct results across all base models are highlighted as first, second, and third, respectively. A † indicates that Qwen-Plus is used as the supervisor for both data curation and evaluation, while a ‡ additionally indicates that the agent used for data curation (or for generating SkillRL's skills) is also Qwen-Plus See Appendix A for a clearer illustration of their differences.

Method	Seen							Unseen							Avg.
	Pick	Look	Clean	Heat	Cool	Pick2	Overall	Pick	Look	Clean	Heat	Cool	Pick2	Overall	
Qwen3.8-Max															
ReAct	94.3	92.3	92.6	93.8	88.0	91.7	92.1	83.3	44.4	80.6	100.0	95.2	82.4	82.1	87.1
Reflexion	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	96.8	100.0	100.0	100.0	99.3	99.7
Qwen-Plus-2507															
ReAct	77.1	76.9	51.9	81.2	80.0	70.8	72.1	91.7	33.3	64.5	69.6	81.0	82.4	70.9	71.5
Reflexion	100.0	92.3	96.3	100.0	96.0	100.0	97.9	100.0	72.2	90.3	91.3	100.0	100.0	92.5	95.2
Qwen3-32B															
ReAct	71.4	84.6	88.9	93.8	84.0	83.3	82.9	91.7	61.1	93.5	95.7	90.5	64.7	85.1	84.0
Reflexion	97.1	100.0	100.0	100.0	92.0	95.8	97.1	100.0	88.9	100.0	100.0	90.5	82.4	94.8	96.0
Qwen3-14B															
ReAct	80.0	92.3	81.5	87.5	64.0	70.8	77.9	91.7	22.2	77.4	95.7	85.7	52.9	73.9	75.9
Reflexion	97.1	92.3	100.0	93.8	84.0	83.3	92.1	100.0	94.4	93.5	100.0	95.2	82.4	94.8	93.5
Qwen3-4B-Instruct-2507															
ReAct	68.6	76.9	63.0	62.5	64.0	50.0	63.6	62.5	66.7	48.4	43.5	38.1	47.1	50.7	57.2
Reflexion	82.9	100.0	85.2	75.0	76.0	79.2	82.1	75.0	88.9	77.4	73.9	57.1	82.4	75.4	78.8
Reflexion†	97.1	76.9	96.3	87.5	84.0	83.3	89.3	91.7	55.6	80.6	78.3	90.5	58.8	77.6	83.5
SFT	62.9	38.5	55.6	68.8	60.0	70.8	60.7	70.8	27.8	61.3	73.9	42.9	11.8	51.5	56.1
SFT + Reflexion	100.0	61.5	96.3	93.8	92.0	79.2	90.0	95.8	38.9	80.7	73.9	90.5	64.7	76.1	83.1
GRPO	97.1	61.5	85.2	75.0	80.0	70.8	81.4	83.3	77.8	87.1	87.0	95.2	88.2	86.6	84.0
GiGPO	91.4	69.2	66.7	81.3	84.0	79.2	80.0	79.2	66.7	54.8	78.3	85.7	70.6	71.6	75.8
SkillRL†	82.9	69.2	96.3	75.0	88.0	87.5	85.0	58.3	94.4	80.6	73.9	81.0	88.2	73.9	79.5
vSFT (one trial)	60.0	84.6	77.8	93.8	56.0	16.7	61.4	75.0	77.8	77.4	91.3	52.4	17.7	67.9	64.7
vSFT	97.1	92.3	96.3	93.8	84.0	91.7	92.9	95.8	88.9	90.3	100.0	100.0	76.5	92.5	92.7 (+13.9)
vSFT†	100.0	100.0	100.0	93.8	88.0	95.8	96.4	100.0	94.4	100.0	100.0	100.0	82.4	97.0	96.7 (+13.2)
vSFT‡	100.0	100.0	96.3	100.0	100.0	100.0	99.3	100.0	100.0	100.0	100.0	100.0	100.0	100.0	**99.7**

4.1 Experimental setup

Benchmarks. Our experiments are conducted on two agent benchmarks: ALFWorld (Shridhar et al., 2021) and WebShop (Yao et al., 2022). ALFWorld is a text-based embodied environment where an agent completes long-horizon household instructions. We report results on both its in-distribution (seen) and out-of-distribution (unseen) splits. WebShop is a simulated e-commerce environment where agents navigate a web interface to find and purchase products satisfying user requirements.

Baselines. We compare our vSFT against the in-context learning counterparts, including ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2023), conventional SFT, and policy-gradient reinforcement learning methods, including GRPO (Shao et al., 2024), GiGPO (Feng et al., 2025), and SkillRL (Xia et al., 2026). All results of the baselines are obtained under a consistent experimental setting.[4]

Implementation details. We use `Qwen3-4B-Instruct-2507` (abbreviated as Qwen3-4B) as the agent and `Qwen-Plus-2507` (abbreviated as Qwen-Plus) either as an alternative supervisor or as both the agent and supervisor to generate hypernetwork training data. The hypernetwork follows the architecture of Doc-to-LoRA (Charakorn et al., 2026) and is initialized with its pretrained weights.[5] It has only 421M trainable parameters, and generates rank-8 LoRA updates applied to the MLP down-projection weights. We finetune the hypernetwork using a learning rate of 4×10^{-6}. For methods that use trial and error, we allow up to 10 trials per task.

4.2 vSFT enables effective self-improvement

Performance across methods. As shown in Tabs. 2 and 3, our vSFT outperforms Reflexion (Shinn et al., 2023), its vRL counterpart, by 13.9% and 18.2% (13.2% and 15.0% with a more capable

[4]We implement RL methods using Agent-R1 (Cheng et al., 2025), so the results may differ substantially from those from Feng et al. (2025).

[5]Directly using the pretrained hypernetwork yields near-zero success rates, highlighting the need for our proposed meta-training in §3.2.

supervisor) in average success rate across ALFWorld's two splits and success rate on Webshop, respectively. Our vSFT also outperforms GRPO by 8.7% and 4.8% on ALFWorld and Webshop, while using less than 4% and 10% of GRPO's training rollout budget, respectively.

Table 3: **Comparisons on WebShop.** We compare vSFT against ICL and RL methods across different base models, reporting average task score and success rate.

Method	Score	SR
Qwen3.8-Max		
ReAct	53.67	36.60
Reflexion	79.16	59.80
Qwen-Plus-2507		
ReAct	51.66	32.80
Reflexion	68.62	47.20
Qwen3-4B-Instruct-2507		
ReAct	40.74	18.80
Reflexion	59.65	30.80
Reflexion†	70.32	44.80
GRPO	65.83	44.20
GiGPO	67.13	50.00
vSFT (one trial)	49.64	24.45
vSFT	73.55	49.00 (+18.2)
vSFT†	82.19	59.80 (+15.0)
vSFT‡	83.58	62.00

Table 4: **Ablation study on supervisor choice.** We evaluate vSFT with different supervisors on WebShop to assess its robustness to its choice.

Supervisor	Score	SR
Qwen		
Qwen3.8-Max	76.47	53.00
Qwen3.7-Plus	76.99	54.40
Qwen-Plus-2507	74.89	50.40
Qwen3-32B	75.89	51.80
Qwen3-4B-Instruct-2507 (baseline)	73.55	49.00
GPT		
GPT-5.6-Sol	77.14	54.00
GPT-5.6-Terra	78.35	55.40
GPT-5.6-Luna	73.81	51.20
DeepSeek		
DeepSeek-V4-Pro	76.54	53.80
DeepSeek-V4-Flash	76.49	53.60
GLM		
GLM-5.2	76.38	52.40
GLM-5.1	78.22	56.00

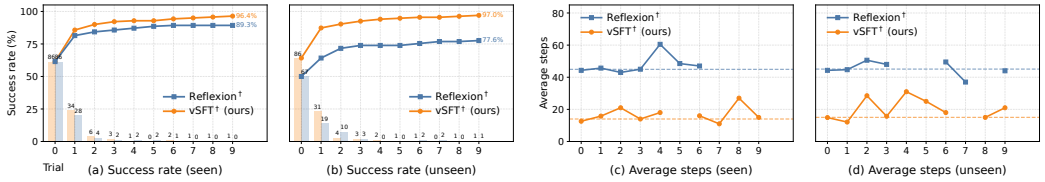

Figure 4: **Success rate and average steps across trials on ALFWorld.** The bars in panels (a) and (b) indicate the number of newly solved tasks per trial. vSFT consistently improves over Reflexion while using shorter contexts and fewer steps, indicating more effective self-improvement and more efficient execution.

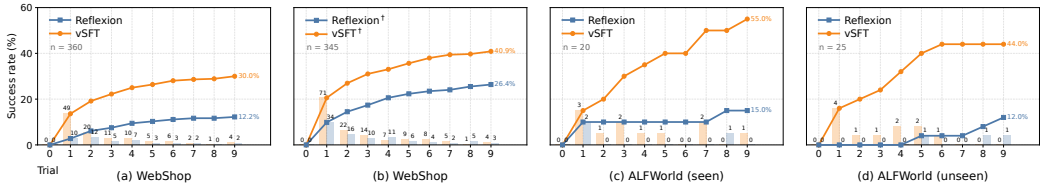

Figure 5: **Success rate on the shared initial failure set across trials.** We compare the self-improvement capabilities of vSFT and Reflexion by tracking their success rates across subsequent trials on the shared task set that both methods fail in the initial trial. The figure indicates the set size as n.

Self-improvement across trials. As shown in Tabs. 2 and 3, our vSFT lags substantially behind most baselines in the initial trial, suggesting that its improved performance stems from more effective self-improvement across subsequent trials. In Fig. 4, we compare our vSFT with Reflexion in terms of success rate and average steps of completed tasks on ALFWorld. We observe that vSFT consistently solves more previously failed tasks per trial while taking fewer average steps, suggesting that the agent improves itself more effectively and decisively via the hypernetwork in response to verbal feedback. We further validate this conclusion in Fig. 5 by directly comparing the success rates across trials on the shared initial failure set. It shows that vSFT still substantially outperforms Reflexion when we control for differences in the initial trial.

4.3 vSFT LEARNS BEYOND THE SUPERVISOR

Performance across models. As shown in Tabs. 2 and 3, when the supervisor is a more capable model, Qwen-Plus, than the agent, Qwen3-4B, our vSFT outperforms the Reflexion baseline that uses Qwen-Plus-2507 as both the agent and supervisor, by 1.5% and 12.6% on ALFWorld and WebShop, respectively. Moreover, when the hypernetwork is further trained solely on the data generated

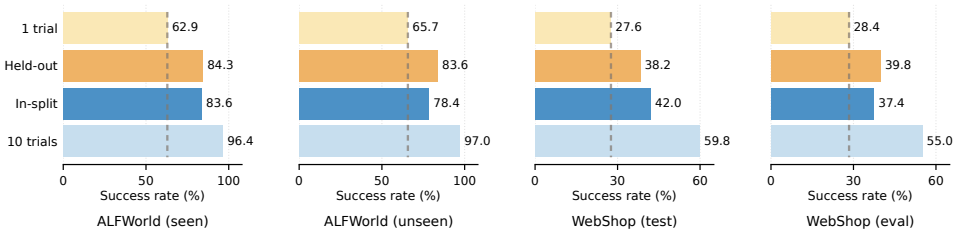

Figure 6: **Task-agnostic and persistent improvement.** Unlike our default setting, where each failed task has a separate LoRA adapter, we adapt the agent using a single LoRA adapter obtained by merging task-specific adapters collected from one of four splits: ALFWorld's seen and unseen splits and WebShop's evaluation and test splits. The adapted agent is evaluated either on the same split used for merging (in-split) or on the other split of the same benchmark (held-out). We include standard vSFT with 1 trial as a baseline and 10 trials as an upper bound. All results are obtained under the vSFT† setting.

by Qwen-Plus, vSFT substantially outperforms this baseline, performs on par with Qwen3.8-Max on ALFWorld, and even surpasses it by 2.2% on WebShop.

Supervisor compatibility. As discussed in §3.1, compared to OPD (Li et al., 2026a), our vSFT is compatible with closed-source API models, as it merely requires feedback in natural language, which may also make it robust to supervisor choice. In Tab. 4, we evaluate vSFT with supervisors from four distinct model families and observe that its performance can be further improved with more capable supervisors, even when the agent and supervisor belong to different model families. Nonetheless, we also find that a more capable supervisor does not always yield better performance.

4.4 vSFT YIELDS PERSISTENT AND GENERALIZABLE INTERNALIZATION

Cross-task improvement. Our vSFT currently provides specific verbal feedback for each failed task and thus generates separate LoRA adapters. Although these adapters can be viewed as fast weights (Wen et al., 2026; Zhang et al., 2026c) or managed using certain infrastructure (Lab et al., 2026), we investigate the potential of using a single model across all tasks that achieves persistent and generalizable self-improvement. In Fig. 6, we adapt the agent using a LoRA adapter that merges adapters collected from one split and evaluate its performance on another split. The results show that this single adapted model still substantially outperforms the one-trial vSFT baseline, suggesting that internalized verbal feedback can generalize to new tasks in a parametric form. By contrast, additional results in Appendix B indicate that such improvements are difficult to achieve through in-context verbal feedback.

Training robustness. As discussed in §3.2, the hypernetwork is trained to learn how to adapt the agent rather than directly produce an adapted model. We examine whether this meta-training process preserves generalization under extended and excessive training. As shown in Fig. 7, although the training loss consistently decreases over additional epochs, vSFT's success rates do not collapse on either validation split, even on the out-of-distribution split. The hypernetwork thus shows no signs of severe overfitting within the evaluated training epochs.

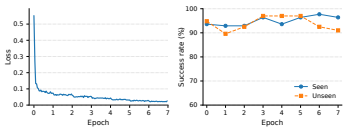

Figure 7: **Training loss and success rates across epochs.** We train the hypernetwork on ALFWorld's training split (left panel) and evaluate on the two validation splits (right panel).

5 CONCLUSION AND FUTURE WORK

We have proposed a novel test-time adaptation method for LLM agents, vSFT, which provides a learnable and effective self-improvement mechanism by iteratively internalizing verbal feedback in a closed loop. We have conducted extensive experiments on ALFWorld and WebShop. The empirical results confirmed that a trained hypernetwork could effectively internalize verbal feedback into an agent, yielding significantly improved performance. Nevertheless, in the current vSFT framework, the hypernetwork that generates LoRA adapters still remains a separate component from the agent. Thus, exploring native self-improvement models that directly generate fast weights to adapt themselves for potentially improving both efficiency and accuracy is worth future work.

AI USE STATEMENT

In this work, we used LLMs as decision-making policies for agents and as supervisors providing verbal feedback in vSFT, as described in §3.1. We also used LLMs to generate hypernetwork training data, as described in §3.2. Additionally, we used LLMs to generate the agent, supervisor, and hypernetwork icons in Fig. 2 and to check our writing for grammatical errors. We did not use LLMs to assist with the design of our core framework or to directly generate manuscript sections. We have reviewed all LLM-assisted aspects of this work and take responsibility for its final content.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *ICLR*, 2024.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-LoRA: Instant transformer adaption. In *ICML*, 2025.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-LoRA: Learning to instantly internalize contexts. In *ICML*, 2026.

Howard Chen, Noam Razin, Karthik Narasimhan, and Danqi Chen. Retaining by doing: The role of on-policy data in mitigating forgetting. *ICML*, 2026.

Mingyue Cheng, Jie Ouyang, Shuo Yu, Ruiran Yan, Yucong Luo, Zirui Liu, Daoyu Wang, Qi Liu, and Enhong Chen. Agent-R1: Training powerful LLM agents with end-to-end reinforcement learning. *arXiv preprint arXiv:2511.14460*, 2025.

Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. GPG: A simple and strong reinforcement learning baseline for model reasoning. In *ICLR*, 2026.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *NeurIPS*, 2025.

Yilin Geng, Haonan Li, Honglin Mu, Xudong Han, Timothy Baldwin, Omri Abend, Eduard Hovy, and Lea Frermann. Control illusion: The failure of instruction hierarchies in large language models. In *AAAI*, 2026.

Liliana Hotsko, Yinxi Li, Yuntian Deng, and Pengyu Nie. Code2LoRA: Hypernetwork-generated adapters for code language models under software evolution. *arXiv preprint arXiv:2606.06492*, 2026.

Pengcheng Jiang, Jiacheng Lin, Zhiyi Shi, Zifeng Wang, Luxi He, Yichen Wu, Ming Zhong, Peiyang Song, Qizheng Zhang, Heng Wang, et al. Adaptation of agentic AI. *arXiv preprint arXiv:2512.16301*, 2025.

Jeonghye Kim, Sojeong Rhee, Minbeom Kim, Dohyung Kim, Sangmook Lee, Youngchul Sung, and Kyomin Jung. ReflAct: World-grounded decision making in LLM agents via goal-state reflection. In *EMNLP*, 2025.

Mind Lab, Song Cao, Vic Cao, Andrew Chen, Kaijie Chen, Cleon Cheng, Steven Chiang, Kaixuan Fan, Hera Feng, Huan Feng, et al. MinT: Managed infrastructure for training and serving millions of LLMs. *arXiv preprint arXiv:2605.13779*, 2026.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huan-ang Gao, Wenkai Yang, Zhiyuan Liu, et al. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026a.

Yibo Li, Zijie Lin, Ailin Deng, Xuan Zhang, Yufei He, Shuo Ji, Tri Cao, and Bryan Hooi. Just-in-time reinforcement learning: Continual learning in LLM agents without gradient updates. *ICML*, 2026b.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. SHINE: A scalable in-context hypernetwork for mapping context to LoRA in a single pass. *arXiv preprint arXiv:2602.06358*, 2026.

Zhengxi Lu, Zhiyuan Yao, Zhuowen Han, Zi-Han Wang, Jinyang Wu, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, et al. Self-distilled agentic reinforcement learning. *arXiv preprint arXiv:2605.15155*, 2026.

Dong Nie. Post-training is about states, not tokens: A state distribution view of SFT, RL, and on-policy distillation. *arXiv preprint arXiv:2605.22731*, 2026.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. *ICML*, 2026.

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.

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. RL's razor: Why online reinforcement learning forgets less. In *ICLR*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

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

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, et al. Skill-SD: Skill-conditioned self-distillation for multi-turn LLM agents. *arXiv preprint arXiv:2604.10674*, 2026.

Qishuai Wen, Zhiyuan Huang, Xianghan Meng, Wei He, and Chun-Guang Li. Mixture-of-top-k attention: Efficient attention via scalable fast weights. *arXiv preprint arXiv:2602.01219*, 2026.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. TIES-merging: Resolving interference when merging models. In *NeurIPS*, 2023.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

Aofan Yu, Chenyu Zhou, Tianyi Xu, Zihan Guo, Rong Shan, Zhihui Fu, Jun Wang, Weiwen Liu, Yong Yu, Weinan Zhang, et al. LatentSkill: From in-context textual skills to in-weight latent skills for LLM agents. *arXiv preprint arXiv:2606.06087*, 2026.

Chenchen Zhang. From reasoning to agentic: Credit assignment in reinforcement learning for large language models. *arXiv preprint arXiv:2604.09459*, 2026.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhong-Zhi Li, Xiangyuan Xue, Yijiang Li, Yifan Zhou, Yang Chen, Chen Zhang, Yutao Fan, Zihu Wang, Songtao Huang, Francisco Piedrahita Velez, Yue Liao, Hongru Wang, Mengyue Yang, Heng Ji, Jun Wang, Shuicheng Yan, Philip Torr, and Lei Bai. The landscape of agentic reinforcement learning for LLMs: A survey. *TMLR*, 2026a.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. MemRL: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026b.

Tianyi Zhang and Zhonghao Qi. Skill-to-LoRA: From using skills to learning behaviors for token-efficient LLM agents. *arXiv preprint arXiv:2606.16769*, 2026.

Yifan Zhang, Steve Ta, Jasper Zhang, Jichen Feng, Shuzhen Li, Yongxin Zhang, Yifeng Liu, Huizhuo Yuan, Mengdi Wang, Quanquan Gu, et al. Fast weight attention for continual learning. *arXiv preprint arXiv:2608.27763*, 2026c.

Yunxiang Zhang, Yiheng Li, Ali Payani, and Lu Wang. AdaMEM: Test-time adaptive memory for language agents. In *ICML*, 2026d.

Hao Zhao, Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. Is in-context learning sufficient for instruction following in LLMs? In *ICLR*, 2025.

Adam Zweiger, Jyo Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. In *NeurIPS*, 2026.

Appendix

A Additional Implementation Details

vSFT variants. As shown in Tab. 2, we implement three vSFT variants that differ in the agent and supervisor models used during data curation and evaluation. For clarity, we summarize their differences in Tab. 5.

Table 5: Agent and supervisor models across vSFT variants during data curation and evaluation.

Method	Data Curation		Evaluation	
	Agent	Supervisor	Agent	Supervisor
vSFT	Qwen3-4B	Qwen3-4B	Qwen3-4B	Qwen3-4B
vSFT†	Qwen3-4B	Qwen-Plus	Qwen3-4B	Qwen-Plus
vSFT‡	Qwen-Plus	Qwen-Plus	Qwen3-4B	Qwen-Plus

LoRA merging. For the experiments in Fig. 6, we merge the effective weight updates (defined in Eq. 7) of all LoRA adapters collected from successful tasks using a weighted variant of TIES-Merging (Yadav et al., 2023). We assign equal total weight to each of the six task categories in ALFWorld and equal weight to each unique verbal feedback document in WebShop.

Rollout cost. For data curation, we collect the trajectories described in Eq. 9 from the training splits of ALFWorld and WebShop under the Reflexion (i.e., vRL) framework. In total, this results in 10,712 rollouts on ALFWorld and 86,533 rollouts on WebShop. For comparison, the default GRPO training recipe in Agent-R1 Cheng et al. (2025) requires 284,160 rollouts on ALFWorld and 926,720 rollouts on WebShop.[6] Our data curation therefore requires only 3.8% and 9.3% of GRPO's rollout budgets on ALFWorld and WebShop, respectively.

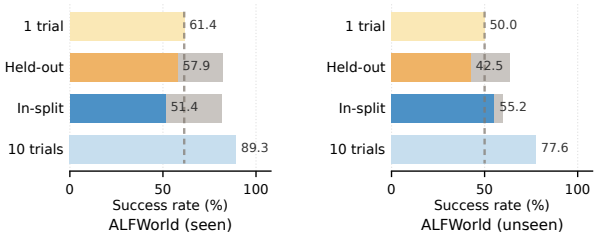

Figure 8: **Persistent skill bank across tasks.** We use Qwen-Plus to summarize task-specific verbal feedback into a persistent skill bank for all tasks. We evaluate the agent with this bank on either the same split used to construct it (in-split) or the other split (held-out). We include Reflexion with 1 trial as a baseline and with 10 trials as an upper bound. These results are obtained under the Reflexion† setting.

B Additional Experimental Results

In-context transfer. In Fig. 6, we show that internalized task-specific verbal feedback can be consolidated through LoRA merging into a persistent, task-agnostic parametric improvement to the agent (in-parameter transfer). We now investigate whether in-context verbal feedback can achieve a similar effect by having an LLM summarize it into a skill bank that the agent persistently retains in context across tasks. As shown in Fig. 8, this in-context counterpart struggles to outperform the baseline and yields substantially smaller gains over the baseline than its in-parameter counterpart, with the gaps highlighted by the grey bars.

Additional performance comparisons. In Tab. 6, we further compare with several methods whose ALFWorld results are only partially reported and thus cannot be included in the comprehensive comparison in Tab. 2.

[6] These results are computed as $\lfloor N/16 \rfloor \times 16 \times 8 \times 10$, where N denotes the training-set size (3,553 for ALFWorld and 11,587 for WebShop), 16 is the batch size, 8 is the group size, and 10 is the number of epochs.

Table 6: **Additional comparisons on ALFWorld.** MemRL and AdaMEM are two methods that enhance the agent through external memory, while ReflAct does so through real-time reflections.

Method	Seen	Unseen
GPT-5-mini		
MemRL (Zhang et al., 2026b)	97.9	–
GPT-4o		
ReflAct (Kim et al., 2025)	–	93.3
Qwen3-8B		
LatentSkill (Yu et al., 2026)	74.3	69.4
Qwen3-4B-Instruct-2507		
AdaMEM-MAX (Zhang et al., 2026d)	65.5	–
vSFT	92.9	92.5
vSFT‡	99.3	100.0

C AGENT SYSTEM PROMPTS

We provide the system prompts used by the agent on ALFWorld and WebShop below. As shown, our prompts are detailed and informative, providing sufficient guidance to the agent and thereby allowing the in-context baselines to realize their full potential.

Prompt C.1: ALFWorld system prompt

```
You are an ALFWorld agent. You interact with a household environment one step at
    a time.

STRICT OUTPUT RULES (you must follow every rule):
1. Output EXACTLY ONE line per turn. Never output multiple lines.
2. Your line must be ONE of the following:
   - A brief thought:  think: <your reasoning>
   - A valid ALFWorld action (see the complete list below)
3. NEVER output observations, environment feedback, or results. Only the
    environment provides observations - you never write them.

COMPLETE LIST OF VALID ACTIONS:
  go to {receptacle}              - move to a receptacle/location
  open {receptacle}               - open a closed container
  close {receptacle}              - close an open container
  take {object} from {receptacle} - pick up an object from a receptacle
  move {object} to {receptacle}   - place an object into/onto a receptacle
  put {object} in/into {target}   - place an object into another object or
    receptacle
  heat {object} with {receptacle} - heat an object using a microwave
  cool {object} with {receptacle} - cool an object using a fridge
  clean {object} with {receptacle} - clean an object using a sinkbasin
  slice {object} with {knife}     - slice an object using a knife
  use {object}                    - toggle an object on/off (e.g. desklamp)
  examine {receptacle or object}  - examine a receptacle or object in detail
  look                            - observe the current surroundings
  inventory                       - check what you are currently carrying

UNDERSTANDING ENVIRONMENT FEEDBACK:
- "OK." after a think: line means your thought was recorded. It is NOT asking
    you to repeat or elaborate - just proceed with your next action.
- "Nothing happens." means your action was invalid (wrong syntax, missing
    preconditions, or targeting a nonexistent object). When you see this, try a
    different action instead of repeating the same one.
```

Prompt C.2: WebShop system prompt

```
You are a WebShop agent. Your goal is to find and purchase a product that
    perfectly matches a given instruction on a simulated e-commerce website.

VALID ACTIONS:
  search[keywords]  - search for products
  click[target]     - click any [bracketed item] visible on the current page.
    This includes product ASINs, option values, navigation buttons, and sub-page
    links.
  think[reasoning]  - record your internal reasoning (environment replies OK.)

SEARCH RULE (CRITICAL):
  search[keywords] is ONLY valid on the initial page (the page that shows
    "[Search]" and the instruction).
  If you are on ANY other page (search results, item page, or sub-page), you
    MUST first click[Back to Search] to return to the initial page before
    issuing search[].
  Issuing search[] from a non-initial page will result in "Invalid action!" and
    waste a step.

ENVIRONMENT CONSTRAINTS:
- Search results show at most 3 products per page. Use [Next >] and [< Prev] to
    browse additional pages if the first page has no suitable match.
- Products may have selectable options (e.g. color, size, flavor). Select all
    relevant options before clicking [Buy Now]; missing options will reduce your
    score.
- Use [< Prev] to navigate back from a sub-page to the item page, or from the
    item page to search results.

OUTPUT FORMAT:
1. Output EXACTLY ONE line per turn.
2. NEVER output observations or environment feedback - only the environment
    provides those.
```