FAST AND SLOW ONLINE LEARNING FOR AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents are typically deployed as static models, yet their deployment conditions can differ from training and continue to evolve over time, requiring agents to continually adapt after deployment. Unlike offline post-training, however, every intermediate policy during deployment serves subsequent queries, making test-time learning essentially an *online learning* problem, where the learner is evaluated by its cumulative performance over the interaction stream rather than only its final performance. This cumulative objective requires both fast adaptation and strong long-run improvement. We therefore introduce *Fast and Slow Online Learning (FSOL)*, a framework that combines memory for fast adaptation with RL for progressively internalizing experience into model weights. We evaluate FSOL on writing-style adaptation, search-augmented QA, WebShop, and ALFWorld across stationary streams, distribution shifts, mixed tasks, and sequential tasks. Across these settings, FSOL consistently achieves higher cumulative performance than memory or RL alone, rapidly adapts to distribution shifts, effectively learns from heterogeneous interleaved tasks, and retains previously acquired capabilities while adapting to new ones.

1 INTRODUCTION

Large language models (LLMs) are typically trained on fixed, pre-collected datasets and then deployed as static models (Suzgun et al., 2026; Tack et al., 2024; Acikgoz et al., 2026). However, deployment conditions can differ from those anticipated during training and continue to evolve over time: websites can be redesigned, app layouts can be refactored, recurring task patterns can emerge, and user preferences can change. Agents therefore need to adapt continually to their deployment environment rather than remain fixed until the next round of offline retraining. Importantly, deployment itself provides a natural source of data for such adaptation: as agents interact with users and environments, they receive queries and feedback that reveal user preferences, task patterns, and properties of the environment (Ouyang et al., 2026b; Suzgun et al., 2026; Zhang et al., 2026b; Ouyang et al., 2026a). This motivates learning from experience online as it arrives.

However, test-time learning has a fundamentally different objective from standard offline training. In offline training, learning takes place on a fixed dataset, and performance is measured on the final checkpoint; the quality of intermediate checkpoints therefore does not directly affect users. In test-time learning, by contrast, every interaction affects a user. The natural objective is therefore cumulative performance over the interaction stream (see the left panel of Figure 1), which rewards both rapid adaptation and strong performance as learning progresses. This perspective makes learning from experience an *online learning* problem (Hazan, 2016; Cesa-Bianchi et al., 2006), where the learner is evaluated throughout the interaction stream rather than only the last checkpoint.

Viewed through this online learning lens, the two existing approaches to learning from deployment experience face different limitations: (1) *Memory* provides a natural mechanism (Wang et al., 2025; Ouyang et al., 2026b; Zheng et al., 2024; Shen et al., 2026; Suzgun et al., 2026; Zhang et al., 2026b): experience is stored in an external memory module and can be immediately retrieved to guide future queries. However, as a form of in-context learning (ICL), its effectiveness depends on the model's ICL capability (Brown et al., 2020; Wei et al., 2022), and it may more readily capture surface-level patterns (e.g., writing styles) than induce fundamental changes in model behavior (Akyürek et al., 2025; Han et al., 2024), such as long chain-of-thought reasoning (Guo et al., 2025). (2) Concurrent work has begun to explore *reinforcement learning (RL) for test-time learning* (Hou et al., 2026; Liu et al., 2026). By updating model weights, RL may provide stronger learning capacity than mem-

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

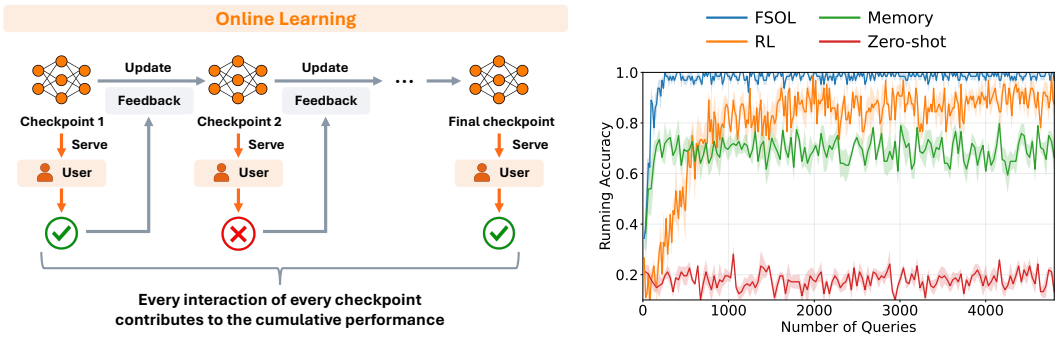


Figure 1: **Left.** Schematic of online learning, where every intermediate checkpoint is deployed to serve users. Therefore, an online learner should be evaluated by its cumulative performance across every interaction. **Right.** Running accuracy of memory, RL, and FSOL. The RL algorithm is REINFORCE++, the memory format is the previous successful responses, and FSOL combines them together. FSOL improves as rapidly as memory in the early stage while eventually reaching high performance even better than RL, resulting in the best cumulative performance.

ory, but often requires many rollouts before performance improves. This is particularly problematic during deployment, where each user query typically provides only one feedback signal (Hou et al., 2026; Liu et al., 2026): a shopping agent cannot place eight orders for one query, nor can an assistant ask a user to rate eight responses. RL must therefore learn from a stream of single-rollout interactions, making slow adaptation costly to cumulative performance. This problem is further amplified under distribution shift (Lu et al., 2018; Cesa-Bianchi et al., 2006; Hazan, 2016), where previously learned behavior may be outdated. Slow adaptation would repeatedly incur performance loss.

Maximizing cumulative performance therefore requires both rapid adaptation to new experience and sustained improvement as experience accumulates. We therefore introduce *Fast and Slow Online Learning (FSOL)*, which combines two complementary timescales of adaptation to meet these requirements. Memory serves as the *fast* component: once a successful interaction is observed, it can be immediately retrieved as context to guide subsequent queries. RL serves as the *slow* component, progressively internalizing accumulated experience into model weights for stronger long-term performance. As illustrated in the right panel of Figure 1, FSOL combines the rapid early adaptation of memory with the higher long-run performance of RL, resulting in a larger area under the curve of running accuracy and thus stronger cumulative performance over the interaction stream. Beyond operating at complementary timescales, the two mechanisms can also reinforce each other. Memory improves rollout quality and steers exploration toward high-reward regions, while an improved policy in turn generates more successful trajectories that enrich the memory. Importantly, FSOL is a generic framework agnostic to the choice of memory and RL algorithm, with the combination yielding synergy as long as both components are individually effective.

We conduct extensive experiments across four tasks: writing-style adaptation (based on the prompts from Fan et al. (2018)), search-augmented question answering (Yang et al., 2018; Kwiatkowski et al., 2019), web shopping with WebShop (Yao et al., 2022), and embodied agent tasks on ALFWorld (Shridhar et al., 2021). We evaluate four online stream settings designed to capture different deployment scenarios: stationary streams, distribution shifts, mixed tasks, and sequential tasks. Our results show that (1) FSOL combines rapid adaptation with strong long-run performance, yielding substantially higher cumulative performance than memory or RL alone; (2) FSOL rapidly adapts to distribution shifts arising from changes in query distributions, reward functions, and environments; (3) FSOL remains effective when heterogeneous tasks are interleaved within a single stream; and (4) FSOL effectively learns over sequential task streams, retaining previously acquired capabilities while remaining sufficiently plastic to learn new tasks.

We summarize our contributions as follows:

1. **Online learning paradigm**: We formulate test-time learning as an online learning problem, an underexplored perspective that evaluates agents by cumulative performance over the interaction

stream rather than only their final performance. This objective naturally requires both rapid adaptation and strong long-run performance.

2. **FSOL framework for online learning**: Motivated by this objective, we introduce Fast and Slow Online Learning (FSOL), a generic framework that combines fast adaptation through memory with slow parameter adaptation through RL to improve cumulative performance.
3. **Comprehensive evaluation across diverse settings**: We evaluate FSOL on four benchmarks under four online settings, including stationary streams, distribution shifts, sequential tasks, and mixed-task streams, covering deployment conditions that are largely unexplored in prior work.

2 Related Work

Learning from Deployment Experience. Prior work has explored adapting language models from deployment or interaction experience through SFT (Liang et al., 2024), self-distillation (Ye et al., 2026; Wang et al., 2026b), and recent concurrent work on streaming RL (Hou et al., 2026; Liu et al., 2026), including single-rollout PPO (Hou et al., 2026) and REINFORCE++-style updates (Liu et al., 2026). Our work aligns with this direction but, to our knowledge, is the first to explicitly formulate cumulative performance over the interaction stream as the objective of agent online learning, and decompose the goal into both fast adaptation and strong long-run performance. We further adhere to a single-feedback constraint, where each query provides only one rollout, which is not considered by some prior work (Wang et al., 2026b; Ye et al., 2026). Finally, we study a broader range of online streams, including distribution shifts, mixed tasks, and sequential tasks, which are largely unexplored in prior work (Wang et al., 2026b; Ye et al., 2026; Liu et al., 2026).

Memory for Experience Reuse. Memory is widely used by LLM agents to reuse experience (Wu et al., 2024; Zheng et al., 2024; Wang et al., 2025; Suzgun et al., 2026; Zhao et al., 2024; Fu et al., 2024; Ouyang et al., 2026b; Zhang et al., 2026b), either as raw trajectories (Wu et al., 2024; Zheng et al., 2024) or summarized lessons (Wang et al., 2025; Ouyang et al., 2026b; Zhang et al., 2026b). A related line of work combines memory with agentic RL, using past experience to improve rollouts and update task executors (Wu et al., 2026; Xia et al., 2026; Shi et al., 2026; Tu et al., 2026; Zhang et al., 2026c) or memory curator (Vishe et al., 2026; Ouyang et al., 2026a). Most of these methods target offline training, reporting the scores of final checkpoints on test sets, and often rely on SFT warm starts, pre-generated memory, or memory abstracted by stronger models (Wu et al., 2026; Xia et al., 2026; Vishe et al., 2026), which are not available online. Ouyang et al. (2026a) focuses on the online setting, but the curator is trained in an offline dataset first. Prior work also emphasizes specialized memory mechanisms such as memory management or abstraction (Xia et al., 2026; Tu et al., 2026; Zhang et al., 2026c; Vishe et al., 2026), with several studies finding that directly combining memory and RL is insufficient. In contrast, in the online setting we find that combining memory with RL on the executor works surprisingly well across different memory formats with basic similarity-based retrieval. This suggests that the role and requirements of memory–RL integration differ between offline training and online learning.

3 Preliminary

Test-time learning requires agents to learn from interaction experience on the fly while serving users. Unlike offline training, performance during learning matters since every interaction is user-facing. This motivates cumulative performance over the stream as the learning objective, requiring both rapid adaptation and strong long-run performance. We formalize this setting below.

Problem Setup. We consider an agent that interacts with an environment to complete tasks in a streaming, online fashion. The agent receives a stream of queries

$$x_1, x_2, \ldots, x_N, \qquad x_i \sim p_i(\mathcal{X}), \tag{1}$$

in which each query is observed once at each time step, and the query distribution $p_i(\mathcal{X})$ may vary along the stream. For query x_i, the agent generates one interaction trajectory:

$$\tau_i \sim \pi_{\theta_i}\left(\cdot \mid x_i\right), \tag{2}$$

where the subscript on θ_i indicates that the policy can be updated along the stream. We assume that after the episode completes, we get a binary final reward $r_i = R(x_i, \tau_i)$ from environments, users, or LLM judges. We use uniform credit to every token of the episode. Queries are accumulated into non-overlapping batches of size B ($B \geq 1$). Upon completing a batch $\{(x_i, \tau_i, r_i)\}_{i \in \mathcal{B}_k}$, where k is the batch index, the policy is updated immediately, and the updated policy then generates the trajectories for batch $\mathcal{B}_{k+1}$.

Goal. The goal is to learn continually from the streaming trajectories and their feedback while maximizing cumulative performance over the interaction stream. With binary reward $r_i \in \{0, 1\}$ for query i, we measure the overall accuracy by $\mathrm{Acc}(N) = \frac{1}{N} \sum_{i=1}^{N} r_i$. Equivalently, relative to an oracle that succeeds on every query, maximizing overall accuracy minimizes the cumulative regret $\mathrm{Regret}(N) = \sum_{i=1}^{N} (1 - r_i)$, which in our setting simply counts the number of failed queries over the stream. This cumulative objective is closely related to regret-based evaluation in online learning (Hazan, 2016), as it evaluates the cost incurred throughout learning.

RL with Single Rollout Per Query. A key constraint of test-time learning is that each query provides only one rollout and its corresponding feedback. This rules out group-based methods such as GRPO (Guo et al., 2025) or RLOO (Kool et al., 2019), which estimate advantages from multiple rollouts of the same query. In contrast, both REINFORCE++ (Hu et al., 2025) and PPO (Schulman et al., 2017) can operate under this constraint without within-query comparisons: REINFORCE++ constructs its baseline across different queries in a batch, whereas PPO uses a learned value function to estimate advantages. We instantiate FSOL with both algorithms, demonstrating that the framework is not tied to a particular RL algorithm.

4 METHODS

The cumulative objective of online learning places two demands on adaptation: the agent must improve quickly from early experience while continuing to achieve strong long-run performance as more experience accumulates. Either slow initial improvement or a low asymptotic performance leads to poor cumulative performance over the interaction stream. In this work, we introduce *Fast and Slow Online Learning (FSOL)*, which provides these two capabilities within a unified online learning loop. It maintains an external memory that makes useful experience immediately available to subsequent interactions, while RL progressively internalizes experience into the policy parameters. The framework is agnostic to both the representation of memory and the choice of RL algorithm. If each component provides effective learning on its timescale, FSOL can yield synergy beyond either mechanism alone.

4.1 MEMORY: FAST NON-PARAMETRIC ADAPTATION

FSOL maintains a memory module $\mathcal{M}_i$ that accumulates useful experience along the interaction stream. Given an incoming query x_i, the memory module retrieves relevant past experience and constructs memory content

$$C_i = \textsc{Memory}(\mathcal{M}_i, x_i), \tag{3}$$

where C_i denotes the contextual information provided by memory to the policy. Depending on the memory implementation, C_i may contain trajectories (Zheng et al., 2024), action sequences (Zhang et al., 2026a), distilled workflows (Wang et al., 2025), reasoning hints (Ouyang et al., 2026b), or other representations of past experience. The policy then serves the query conditioned on this memory content,

$$\tau_i \sim \pi_\theta(\cdot \mid x_i, C_i). \tag{4}$$

After receiving feedback r_i, the memory is updated according to its memory-specific update rule,

$$\mathcal{M}_{i+1} \leftarrow \textsc{UpdateMemory}(\mathcal{M}_i, x_i, \tau_i, r_i). \tag{5}$$

Because newly acquired experience can affect subsequent interactions immediately without requiring parameter updates, memory provides the fast adaptation mechanism in FSOL.

FSOL does not assume a particular memory representation or update rule. In our experiments, we instantiate the framework with both raw experience memory (action sequences for agentic tasks and

raw successful responses for the Writing task) and distilled workflow memory. For raw experience memory, the output of each turn is stored directly and retrieved using embedding similarity, without additional LLM calls for experience distillation. For workflow memory, successful experiences are distilled by an LLM into more abstract and reusable procedures following AWM (Wang et al., 2025). In our experiments the model used for distillation is the same as the policy model. Details of the two memory instantiations are provided in Appendix B.1. In both cases, the memory is append-only and does not involve revision, deletion, or merging.

4.2 RL: Slow Parametric Adaptation

Alongside fast contextual adaptation, FSOL uses RL to progressively internalize interaction feedback into the policy parameters. After collecting deployment trajectories, the policy is updated using the rewards obtained from the environment. Unlike memory, whose effect is immediately available through the context, parameter updates accumulate information across interactions and provide a slower but potentially more expressive form of adaptation.

FSOL does not assume a specific RL algorithm. We instantiate the framework with both PPO (Schulman et al., 2017) and REINFORCE++, with both satisfying single-rollout constraint. In both cases, policy optimization operates on trajectories generated under the memory-augmented policy. Importantly, when evaluating the likelihood of a collected trajectory, the policy is conditioned on the same memory content C_i that was available when that trajectory was generated. Specifically, for algorithms using a policy probability ratio, we define the token-level ratio as

$$\rho_{i,l}(\theta) = \frac{\pi_\theta\left(\tau_{i,l} \mid x_i, C_i, \tau_{i,1:l-1}\right)}{\pi_{\theta_{\text{old}}}\left(\tau_{i,l} \mid x_i, C_i, \tau_{i,1:l-1}\right)}, \tag{6}$$

where l indexes generated tokens within trajectory τ_i, and $\pi_{\theta_{\text{old}}}$ denotes the behavior policy that generated the trajectory. The same principle applies to other policy-dependent quantities, such as the KL regularization term: both the current and reference policies are evaluated under the same memory content C_i.

4.3 FSOL: Fast and Slow Online Learning

FSOL integrates the two adaptation timescales through a shared stream of deployment experience. At each interaction, the current memory state determines the context C_i used by the policy to serve the incoming query. The resulting trajectory receives environment feedback, which can immediately update the memory available to future interactions and also serves as the learning signal for subsequent parameter updates. The updated memory and policy are then jointly carried forward through the interaction stream. Algorithm 1 summarizes the procedure.

The two adaptation mechanisms interact throughout learning. Fast contextual adaptation can improve the success rate early in the stream, providing more informative trajectories for parameter learning and mitigating sparse learning signals when the policy itself has not yet adapted. Conversely, as RL improves the policy, it can generate more successful and higher-quality experience, which is incorporated by the memory mechanism and becomes available to subsequent interactions. Thus, rather than operating as two independent learning procedures, contextual and parametric adaptation continuously shape the experience available to one another.

This interaction directly targets the objective of online learning. Fast adaptation reduces errors incurred while parameter learning is still catching up, whereas slow adaptation progressively internalizes experience and can improve beyond what can be achieved through context alone. FSOL therefore aims not merely to obtain a strong final checkpoint, but to maintain high cumulative performance throughout the entire interaction stream.

5 Experiments

5.1 Experimental Setup

Benchmarks. We evaluate on four benchmarks spanning preference-following generation, search-augmented QA, web navigation, and embodied household tasks. A similar writing task is used by

Algorithm 1 Fast and Slow Online Learning (FSOL)

Require: base policy π_{θ_0}, interaction stream $x_1, \ldots, x_N$, batch size B

$\theta \leftarrow \theta_0$
$\mathcal{M}_0 \leftarrow \emptyset$
Partition the stream into consecutive batches $\mathcal{B}_1, \mathcal{B}_2, \ldots$ of batch size B
for each consecutive batch $\mathcal{B}_k$ **do**
    for each $x_i \in \mathcal{B}_k$ **do**
        $C_i \leftarrow \textsc{Memory}(\mathcal{M}_i, x_i)$
        $\tau_i \sim \pi_\theta(\cdot \mid x_i, C_i)$ ▷ single rollout
        $r_i \leftarrow R(x_i, \tau_i)$
        $\mathcal{M}_{i+1} \leftarrow \textsc{UpdateMemory}(\mathcal{M}_i, x_i, \tau_i, r_i)$
    end for
    $\theta \leftarrow \text{RL}\textsc{Update}\,(\theta; \{x_i, C_i, \tau_i, r_i\}_{i \in \mathcal{B}_k})$
end for

Hou et al. (2026) to demonstrate distribution shifts, and the other three agentic benchmarks are generally used by prior agentic RL work (Feng et al., 2026; Xia et al., 2026; Wu et al., 2026; Jiang et al., 2026; Wang et al., 2026a). **Writing** is a single-turn preference learning task: the agent is asked to write a paragraph following the given context, three candidate styles are available, and only one style is rewarded. The given context is from Fan et al. (2018). **Search** evaluates multi-turn tool-augmented question answering with HotpotQA and Natural Questions datasets (Yang et al., 2018; Kwiatkowski et al., 2019), where the agent chooses between a search query and providing an answer at each turn. **WebShop** (Yao et al., 2022) is a web environment in which an agent decides among actions such as issuing a search query, selecting a product, or making a purchase. to get the item described in the instruction. **ALFWorld** (Shridhar et al., 2021) is a household embodied environment where an agent takes actions such as moving between locations, picking up or placing objects, or manipulating objects through actions such as heating, cooling, and cleaning.

Methods. We compare the following methods. **Zero-shot**: the base policy is evaluated independently on each query using the model's recommended sampling parameters; see Appendix A. **Memory**: we consider two memory formats, raw memory and Agent Workflow Memory (AWM). **RL**: we apply REINFORCE++ and PPO to the data stream without memory, corresponding to Algorithm 1 with memory retrieval and insertion disabled. **FSOL**: we consider three combinations of RL algorithms and memory formats: PPO with AWM, PPO with raw memory, and REINFORCE++ with raw memory. For PPO with AWM, we use a frozen copy of the base policy as the external model for distilling successful experience into workflows.

Implementation. We use `Qwen3-4B-Instruct-2507` as the base policy (Yang et al., 2025). The codebase is modified from the `verl-agent` repository (Feng et al., 2026). The AdamW optimizer (Kingma & Ba, 2014; Loshchilov & Hutter, 2019) is used, with a constant learning rate of 1×10^{-6}. We do one gradient update per batch. Each RL run uses 4 NVIDIA H100 GPUs for three agentic tasks, and 5 GPUs for writing tasks since it requires a judge model. We report the average over 4 seeds; shaded regions in figures indicate the standard error of the mean. Other details are shown in Appendix A. The prompts for the agents and judge models are shown in Appendix B.

5.2 Stationary Stream

We first consider stationary streams, where queries are randomly shuffled while the environment and reward remain unchanged.

Table 1 summarizes the overall accuracy across four benchmarks. Across different RL algorithms and memory formats, FSOL consistently improves over the corresponding standalone components. This suggests that the benefit of FSOL is not tied to a particular choice of memory representation or RL algorithm. The only exception is PPO + AWM on WebShop, where AWM itself provides little benefit over the zero-shot policy, and consequently does not improve PPO. In particular, raw memory works very well in every benchmark, and the best method for each benchmark always falls into FSOL with raw memory.

Table 1: Overall accuracy (↑) across four benchmarks. Numbers are percentages (mean ± SEM). The best result in each column is **bolded**, and the second-best is underlined.

	Method	WebShop	Search	ALFWorld	Writing	Average
Zero-shot	Zero-shot	4.0 ± 0.2	15.0 ± 0.0	13.0 ± 0.1	17.9 ± 0.3	12.5
Memory	Raw Memory	34.0 ± 1.0	33.8 ± 0.2	25.0 ± 0.5	68.7 ± 1.9	40.4
	AWM	6.1 ± 0.7	33.8 ± 0.2	20.3 ± 0.6	–	–
RL	REINFORCE++	38.4 ± 5.6	35.8 ± 1.9	33.7 ± 3.8	79.8 ± 0.4	46.9
	PPO	47.7 ± 2.7	35.1 ± 0.6	40.6 ± 3.9	49.7 ± 4.8	43.3
FSOL (Ours)	PPO + AWM	46.7 ± 2.9	37.8 ± 0.2	48.8 ± 3.0	–	–
	PPO + Raw	$\mathbf{56.3 \pm 0.9}$	37.5 ± 0.6	50.9 ± 3.2	96.3 ± 0.5	**60.2**
	RF++ + Raw	52.5 ± 1.5	$\mathbf{38.8 \pm 0.9}$	$\mathbf{52.1 \pm 2.4}$	$\mathbf{97.4 \pm 0.3}$	**60.2**

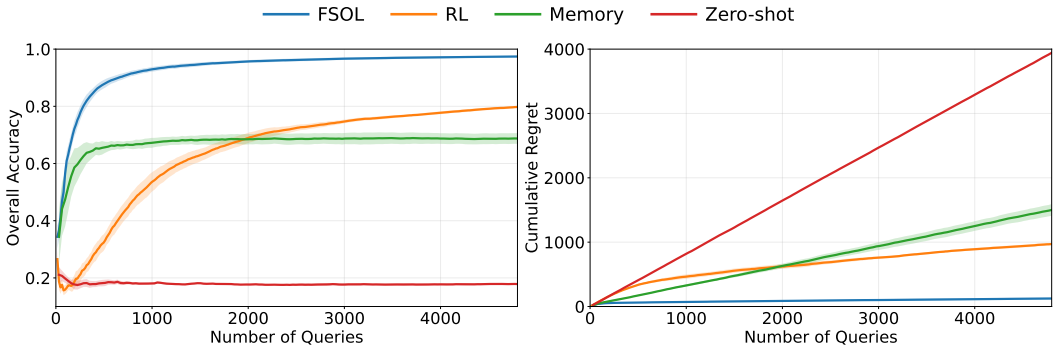


Figure 2: Writing under a stationary stream. The RL algorithm is REINFORCE++, and the memory format is the previous successful responses. **Left:** overall accuracy versus number of queries. **Right:** cumulative regret, the number of failed queries over all queries served so far.

We use Writing to examine the learning dynamics more closely. Figure 1 clearly illustrates the different timescales of memory and RL. Memory adapts almost immediately on Writing but quickly plateaus, whereas RL improves more slowly but eventually reaches a higher ceiling. FSOL follows the rapid initial improvement of memory and continues improving after memory saturates, thereby achieving high performance with fewer failed interactions. Their overall-accuracy curves (left panel of Figure 2) therefore exhibit a clear crossover, with memory performing better early in training and RL eventually surpassing it. FSOL avoids this trade-off, closely tracking the rapid early improvement of memory while continuing to improve beyond its plateau. Consequently, FSOL maintains the highest overall accuracy throughout training. This advantage is also clear in cumulative regret (right panel of Figure 2): FSOL quickly flattens as its accuracy approaches 100%, while memory continues accumulating errors due to its lower ceiling and RL incurs many errors during its slower initial adaptation.

The same pattern generalizes beyond Writing. Across Search, WebShop, and ALFWorld, FSOL combines rapid early improvement with strong long-run performance, leading to the highest overall accuracy in Table 1. Detailed learning curves are provided in Appendix C.

5.3 Distribution Shift

We next evaluate FSOL under abrupt distribution shifts. We consider three qualitatively different changes: reward rules, environment dynamics, and query distributions.

Reward shift. In the Writing task, we reward three different styles at different stages, similar to Hou et al. (2026). To prevent retrieving outdated memory, for FSOL and memory we only retain the memory items generated by recent 64 tasks. The left panel of Figure 3 shows that after each shift,

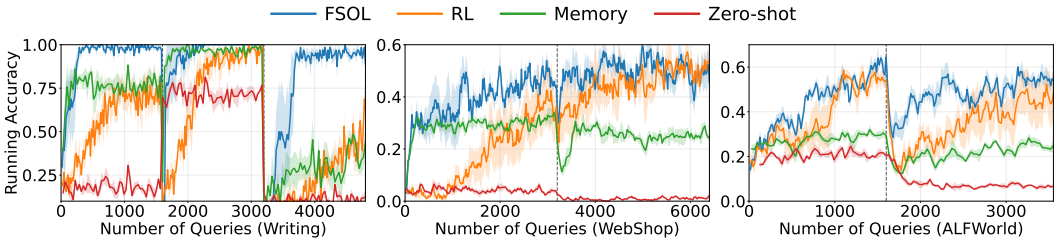


Figure 3: The running accuracy for three benchmarks under distribution shift. The dashed vertical lines show the shift step. The RL algorithm is REINFORCE++, and the memory format is the action sequence from past successful trajectories. **Left:** Writing, where the rewarded style changes at three stages. **Middle:** WebShop, where the purchase methods change from buying directly on product pages to first adding items to the cart and then checking out. **Right:** ALFWorld, where the query distribution changes from the mixture of pick and pick2 task subsets to the other four subsets.

memory recovers fast but reaches a lower ceiling, whereas RL adapts substantially more slowly. FSOL recovers almost immediately and returns to near-perfect accuracy.

Environment shift. In WebShop, we change the purchasing procedure from directly buying an item on its product page to first adding it to the cart and then checking out. For FSOL and memory, we only retain the memory items generated by recent 160 queries. From the middle panel of Figure 3 we can see that FSOL recovers the fastest, and maintains the highest accuracy throughout the training process. Interestingly, RL recovers faster than memory in this setting, indicating that rapid memory adaptation depends on whether previously successful experience remains reusable after the shift.

Query type shift. In ALFWorld, we construct a stream that begins with a mixture of the `pick` and `pick2` subsets and then switches to the mix of the other four subsets. In the right panel of Figure 3, FSOL exhibits a smaller performance drop, recovers more quickly, and consistently maintains the highest running accuracy throughout the stream.

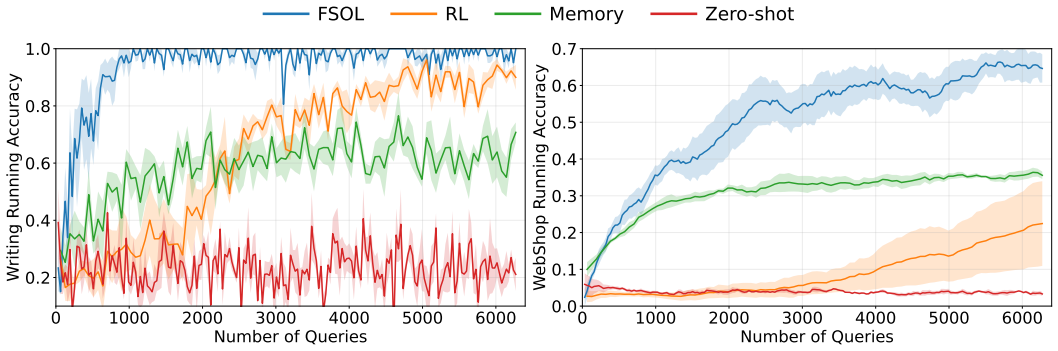


Figure 4: The running accuracy of WebShop and Writing in a mixed stream of two tasks, where there are 4200 WebShop tasks and 2100 Writing tasks. The RL algorithm is PPO, and memory format is raw responses.

5.4 Mixed Task Stream

We further consider a heterogeneous stream in which Writing and WebShop queries are shuffled, with 2,100 Writing and 4,200 WebShop queries. To reduce memory interference from different tasks, we use the full prompt (See Appendices B.4 and B.2) as the retrieval key and only retrieve memory entries whose cosine similarity exceeds 0.35. As shown in Figure 4, FSOL maintains higher running accuracy in both tasks than the standalone approaches throughout the stream, demonstrating

that its fast and slow adaptation mechanisms remain complementary in a multitask stream. We also observe that PPO struggles to learn WebShop despite using a larger batch size. One possible explanation is the difficulty of value estimation in heterogeneous streams: the critic must simultaneously fit tasks with substantially different learning dynamics and reward distributions, which can make value estimates unstable. In contrast, FSOL improves steadily on both tasks despite using a smaller batch size, suggesting that memory can partially alleviate this difficulty.

5.5 Sequential Task Streams

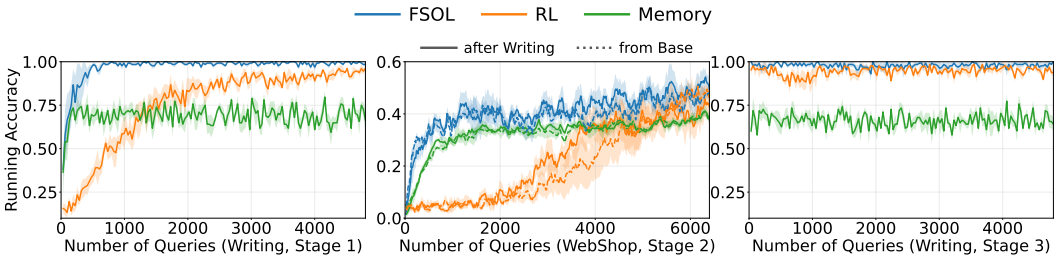


Figure 5: The running accuracy of three stages in a sequential task stream of Writing → WebShop → Writing. In the middle panel, dashed lines denote runs initialized from the base model, while solid lines denote runs initialized from the policy checkpoint and/or memory bank previously adapted to Writing.

Finally, we evaluate a sequential Writing → WebShop → Writing stream to study the plasticity to learn new capabilities and the retention of previously learned capabilities (Dohare et al., 2024). At each transition, the learned policy and/or memory bank is carried into the next stage. Similar to mixed task stream, to reduce cross-task memory interference, we set a cosine-similarity threshold of 0.35 for memory retrieval. Figure 5 shows that memory, RL, and FSOL all retain strong plasticity: when transitioning from Writing (left panel) to WebShop (middle panel), their learning curves (solid lines) closely track the corresponding runs initialized directly from the base model (dashed lines). When returning to Writing after WebShop, performance also remains close to that achieved at the end of the first Writing stage, indicating that the previously acquired capability is largely retained.

6 Conclusion

We study test-time learning of LLM agents from an online learning perspective, where the objective is to maximize cumulative performance over the interaction stream rather than only the performance of the final checkpoint. This perspective motivates Fast and Slow Online Learning (FSOL), a generic framework that combines memory for rapid adaptation with RL for progressively internalizing deployment experience. Across four benchmarks and diverse online settings, including stationary streams, distribution shifts, mixed tasks, and sequential tasks, FSOL consistently achieves stronger cumulative performance than memory or RL alone. These results suggest that combining complementary mechanisms for immediate and long-term adaptation provides an effective foundation for LLM agents that continually learn from deployment experience.

AI Use Statement

In this work, we used generative AI tools to assist with software implementation and to improve the readability and language of the manuscript. In particular, generative AI was used to help write, debug, and refactor code based on specifications provided by the authors, and to edit author-written text for grammar, clarity, and style. We did not use generative AI to formulate the research questions, develop the core methodology, design the main experiments, or interpret the experimental results. All AI-assisted code was reviewed and tested by the authors, and all AI-assisted text was manually reviewed and revised. We take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

Ethics Statement

This work studies methods for improving the online adaptation of large language model agents using existing public benchmarks and datasets. It does not involve human subjects, collection of private or personally identifiable information, or deployment in high-stakes real-world settings. As with general-purpose LLM agents, the proposed methods could potentially be applied in unintended or harmful ways; however, this work focuses on improving learning efficiency and adaptation rather than enabling any specific harmful capability. We encourage responsible use of the proposed methods and careful evaluation before deployment in real-world applications.

Reproducibility Statement

The code to reproduce the results in this paper will be released.

References

Emre Can Acikgoz, Cheng Qian, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. TT-SI: Self-improving LLM agents with test-time training. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 9483–9508, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.462. URL https://aclanthology.org/2026.findings-acl.462/.

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 942–963, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Nicolo Cesa-Bianchi, Gábor Lugosi, et al. *Prediction, learning, and games*, volume 1. Cambridge university press Cambridge, 2006.

Shibhansh Dohare, J Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A Rupam Mahmood, and Richard S Sutton. Loss of plasticity in deep continual learning. *Nature*, 632(8026): 768–774, 2024.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 889–898, 2018.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408, 2026.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems*, 37: 119919–119948, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Pengrui Han, Peiyang Song, Haofei Yu, and Jiaxuan You. In-context learning may not elicit trustworthy reasoning: A-not-b errors in pretrained language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 5624–5643, 2024.

Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325, 2016.

Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. Single-rollout asynchronous optimization for agentic reinforcement learning. *arXiv preprint arXiv:2607.07508*, 2026.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. Reinforce++: Stabilizing critic-free policy optimization with global advantage normalization, 2025. URL https://arxiv.org/abs/2501.03262.

Yulun Jiang, Liangze Jiang, Damien Teney, Michael Moor, and Maria Brbic. Meta-rl induces exploration in language agents. In *International Conference on Learning Representations*, volume 2026, pp. 3279–3303, 2026.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 REINFORCE samples, get a baseline for free!, 2019. URL https://openreview.net/forum?id=r1lgTGL5DE.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Juhao Liang, Ziwei Wang, Zhuoheng Ma, Jianquan Li, Zhiyi Zhang, Xiangbo Wu, and Benyou Wang. Online training of large language models: Learn while chatting. *arXiv preprint arXiv:2403.04790*, 2024.

Linbo Liu, Yuzhe Lu, Youzhi Luo, Sijun Tan, Panpan Xu, and Luke Huan. Continual learning via real-time rl for agents. *rLLM Blog*, July 2026. URL https://rllm-project.com/post.html?post=realtime_rl.md.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering*, 31(12):2346–2363, 2018.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, et al. Skillos: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026a.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Maohao Shen, Kaiwen Zha, Zexue He, Zhang-Wei Hong, Siru Ouyang, Jongha Jon Ryu, Prasanna Sattigeri, Suhas Diggavi, and Gregory W. Wornell. Decocted experience improves test-time inference in LLM agents. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=gAmz0gKgXj.

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 2026.

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

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

Jihoon Tack, Jaehyung Kim, Eric Mitchell, Jinwoo Shin, Yee Whye Teh, and Jonathan Richard Schwarz. Online adaptation of language models with a memory of amortized contexts. *Advances in Neural Information Processing Systems*, 37:130109–130135, 2024.

Songjun Tu, Chengdong Xu, Qichao Zhang, Yaocheng Zhang, Xiangyuan Lan, Linjing Li, Dong Li, and Dongbin Zhao. Dynamic dual-granularity skill bank for agentic rl. *arXiv preprint arXiv:2603.28716*, 2026.

Yash Vishe, Rohan Surana, Xunyi Jiang, Zihan Huang, Xintong Li, Nikki Lijing Kuang, Tong Yu, Ryan A Rossi, Jingbo Shang, Julian McAuley, et al. Skill-r1: Agent skill evolution via reinforcement learning. *arXiv preprint arXiv:2605.09359*, 2026.

Jiawei Wang, Jiacai Liu, Yuqian Fu, Yingru Li, Xintao Wang, Yuan Lin, Yu Yue, Lin Zhang, Yang Wang, and WANG KE. Harnessing uncertainty: Entropy-modulated policy gradients for long-horizon LLM agents. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=YVNn8t1XMX.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. Openclaw-rl: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, and Hung yi Lee. Streambench: Towards benchmarking continuous improvement of language agents. In *NeurIPS Datasets and Benchmarks Track*, 2024.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. Evolver: Self-evolving llm agents through an experience-driven lifecycle, 2026. URL https://arxiv.org/abs/2510.16079.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. In *ICLR 2026 Workshop on Memory for LLM-Based Agentic Systems*, 2026. URL https://openreview.net/forum?id=By7Pj576U3.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui

Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

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.

Tianzhu Ye, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. Online experiential learning for language models. *arXiv preprint arXiv:2603.16856*, 2026.

Dylan Zhang, Yanshan Lin, Zhengkun Wu, Yihang Sun, Bingxuan Li, Dianqi Li, and Hao Peng. Useful memories become faulty when continuously updated by llms. *arXiv preprint arXiv:2605.12978*, 2026a.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=eC4ygDs02R.

Xiaoying Zhang, Zichen Liu, Yipeng Zhang, Xia Hu, and Wenqi Shao. Retroagent: From solving to evolving via retrospective dual intrinsic feedback. *arXiv preprint arXiv:2603.08561*, 2026c.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *International Conference on Learning Representations*, volume 2024, pp. 19036–19066, 2024.

A Hyperparameter and settings

Table 2 lists the hyperparameters shared by environments, Table 3 lists the hyperparameters about PPO, and Table 4 the per-environment settings. All values are those actually used by the launch scripts; where a value is not listed it is the framework default (Feng et al., 2026). For training-free evaluation, including both zero-shot and memory, we use the recommended sampling parameters, where Temperature=0.7, TopP=0.8, and TopK=20 [1].

Turn-level versus episode-level normalization in REINFORCE++. We find that episode-level reward normalization performs poorly on ALFWorld. Instead, we assign the final episode reward to every turn and compute the reward mean and standard deviation over all turns in the batch, which empirically performs better. We therefore use turn-level normalization for ALFWorld. For Search, WebShop, and Writing, we use the original REINFORCE++ formulation: each episode is treated as one sample, and the mean and standard deviation are computed over episode rewards.

Approximating sequential rollouts with batched rollouts. In a strictly online setting, queries arrive sequentially and each rollout is generated before observing the next query. Simulating this process exactly is computationally expensive, as it prevents parallel rollout generation. We therefore parallelize rollouts whenever doing so does not materially change the learning process. For zero-shot evaluation, parallelization is exactly equivalent to sequential generation. For RL, all rollouts

[1] https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507

Table 2: Settings shared across all environments and all learned methods.

Setting	Value
Base policy	Qwen3-4B-Instruct-2507
Fine-tuning type	full parameters
Embedding model (retrieval)	Qwen3-Embedding-4B
Retrieved items per query	top-3
Rollouts per query	1
Gradient updates per batch	1 (no mini batch)
Optimizer	Adam
Adam (β_1, β_2)	$(0.9,\ 0.999)$
Weight decay	0.01
Gradient clipping	1.0
Learning rate	1×10^{-6}
KL Coefficients (FSOL)	0.05
KL Coefficients (RL)	0.01
Batch size (FSOL with REINFORCE++ and REINFORCE++ alone)	16
Batch size (FSOL with PPO)	32
Batch size (PPO)	64
lr schedule	constant (no warmup, no decay)
PPO clipping range	$[0.8,\ 1.2]$
KL loss	enabled, `low_var_kl` against π_{θ_0}
Entropy coefficient	0
Max prompt length	4096
Rollout temperature	1.0
Rollout TopP	1.0
Rollout TopK	N/A
Rollout engine	vLLM
GPUs for RL training	4
Tensor parallelism	4

Table 3: Additional settings for the PPO variant.

Setting	Value
Discount γ	1.0
GAE λ	1.0
Critic initialisation	Qwen3-4B-Instruct-2507 (same as actor)
Critic learning rate	1×10^{-5}
Critic lr schedule	constant (no warmup, no decay)
Critic weight decay	0.01
Critic gradient clipping	1.0
Critic value clipping range	0.5
Critic epochs per batch	1
Critic warmup steps	0
Gradient checkpointing (critic)	enabled
Critic parameter offload	disabled
Critic optimizer offload	enabled

within an update batch are generated by the same policy before the parameter update, so generating them in parallel does not change the algorithm.

Memory-based methods require more care because newly collected experience can immediately affect subsequent queries. Under fully sequential rollouts, a successful trajectory from query i can be added to memory and retrieved by query $i+1$; under batched rollouts, queries within the same batch instead share the memory state available at the beginning of the batch. This distinction is particularly important when relevant memory is sparse: at the beginning of the stream, or immediately after a distribution shift, even a single successful trajectory can substantially improve subsequent rollouts and trigger a compounding improvement in memory quality. We therefore preserve fully sequential

Table 4: Per-environment settings. N is the stream length and B the online batch size. The KL coefficient is the one value we set differently for FSOL than for pure RL: conditioning on retrieved exemplars shifts the rollout distribution away from π_{θ_0}, and a larger penalty keeps the update stable. For the memory within each episode, we do not feed all the past turns into the model. Instead, we only retain the recent 4 observations for WebShop, and recent 4 observation-action pairs for ALFWorld and Search.

	Writing	**WebShop**	**ALFWorld**	**Search**
Stream length N	4800	6400	3553	6400
Max env. steps per episode	1 (single-turn)	15	50	4
Max response length	1024	256	256	512
Observation history length	–	4	4	4

rollouts for the first 320 queries for Writing, WebShop, and Search-augmented QA, their corresponding distribution shift and sequential setting, and also the mixed task setting. For ALFWorld and its distribution shift setting, we do one-by-one rollout for the first 640 queries. For distribution shift in Sec. 5.3, since we just retrieve recent 10 batches or 2 batches for ALFWorld and Writing, respectively, so we just do on-by-one rollout on the 2 or 10 batches after starting or shift, which are 64 samples and 160 samples, respectively. After this warm-up period, the memory module contains sufficiently many relevant successful trajectories that the marginal effect of adding an individual trajectory is much smaller, and we switch to batched rollout generation for computational efficiency.

Reward details. WebShop and ALFWorld use the environment's own success flag. Search awards 1 when the final answer matches the ground truth. These three environments have been implemented in `verl-agent` codebase (Feng et al., 2026). On Writing, an LLM judge (`Qwen2.5-7B-Instruct` (Qwen et al., 2025)) reports a style label and a quality verdict, and the reward is the product of the two, same as (Hou et al., 2026). Credit requires both matching the hidden target style and passing the quality check, in order to avoid overly fit to the desired styles with meaningless words. The judge prompt is shown in Appendix B. All four rewards are therefore binary.

B PROMPTS

This appendix lists the prompt templates used for the four environments, exactly as they appear in our implementation. Placeholders in braces are filled in at run time: {`task_description`} is the task or query, {`current_observation`} the current environment observation, {`available_actions`} / {`admissible_actions`} the admissible action set, and {`step_count`}, {`current_step`}, {`history_length`}, {`action_history`} and {`memory_context`} the within-episode interaction history. For the multi-turn environments (WebShop, ALFWorld, Search) we show both the first-step template, used when no within-episode history exists yet, and the template used at all later steps.

The memory-related prompt is shared by all four tasks and is therefore given once, in Appendix B.1, rather than repeated per task. For memory-augmented methods (Memory and FSOL), the retrieved memory block is *prepended* to the task template of whichever task is running, separated from it by a blank line, so that the policy context is always "memory block + task prompt".

B.1 MEMORY

A retrieved memory block consists of a one-paragraph header followed by the K retrieved items, in descending order of cosine similarity, separated by blank lines. The header is the only instruction text about memory that the policy ever receives; the items themselves are the raw stored experience (Section 4.1) and contain no additional instructions.

Memory block: header (WebShop, ALFWorld, Search)

```
Below are some memory items distilled from your past experience on
similar tasks. Use them as reference when they are relevant to the
current task.

{memory_item_1}

{memory_item_2}
...
```

For the three agentic tasks each item is the query of a past successful episode followed by the action sequence that solved it, in the following format.

Memory item format (WebShop, ALFWorld, Search)

```
Query: {past_task_description}
Actions:
Step 1: {action_1}
Step 2: {action_2}
...
```

The Writing task is single-turn and has no action sequence, so an item is a past writing prompt together with the response that was rewarded for it, and the header is worded accordingly. Examples are renumbered $1, \ldots, K$ when several items are concatenated into one block.

Memory block: header and item format (Writing)

```
Below are some memory items distilled from your past experience on
similar writing prompts. Use them as reference when they are relevant
to the current writing prompt.

Example Query 1: {past_writing_prompt}

Example Response 1: {past_passage}

Example Query 2: {past_writing_prompt}

Example Response 2: {past_passage}
...
```

AWM. The formats above are those of Synapse, which stores the raw episode text verbatim. AWM instead calls the memory LLM once per *successful* episode to induce reusable workflows from it, and stores the model's output verbatim as the item. The block header and the retrieval procedure are unchanged — only the item content differs. An AWM item is therefore one or more named workflows, each a bulleted sub-routine in which the task-specific parts are replaced by descriptive variable names in angle brackets, so the workflow transfers to future episodes whose literal values differ.

Memory item format (AWM)

```
Workflow 1: {short_name_of_workflow}
- {action step 1, with non-fixed elements written as <variable_name>}
- {action step 2, ...}

Workflow 2: {short_name_of_workflow}
- {action step 1, with non-fixed elements written as <variable_name>}
- {action step 2, ...}
...
```

The workflows are induced by the prompt below, issued as the system message of a single chat completion at temperature 0.7. The user message is the finished episode, rendered in the same `Query / Actions / Step` i format as a Synapse agentic item above (for the Writing task, in

the `Example Query / Example Response` format instead). Failed episodes are discarded and never reach this prompt; if the call fails or returns empty text, no item is written for that episode.

Workflow induction prompt (AWM), system message

```
Given a successful agent trajectory, your task is to extract the
common, reusable workflows used to solve the task.
The trajectory contains a natural language query and a series of
actions that solved the task. You need to find meaningful sub-routines
of actions and extract each of them as a workflow.
Each workflow should be a commonly-reusable sub-routine. Do not
generate similar or overlapping workflows. Each workflow should have
at least two steps. Represent the non-fixed elements (e.g. a search
keyword, a clicked item name) with descriptive variable names rather
than the literal value, so the workflow generalizes to future tasks.

## Output Format
Your output must strictly follow the format shown below:

Workflow i: <short name of the workflow>
- <action step 1, with non-fixed elements written as <variable_name>>
- <action step 2, ...>
```

B.2 WRITING

The writing task is single-turn, so there is only one template. The three candidate styles are listed, but which style is rewarded is deliberately not revealed: that is what the agent has to discover from feedback, and re-discover after each distribution shift.

Writing: policy prompt

```
You are a creative writer. Continue the story for the writing prompt
below.

Writing prompt: {task_description}

Write your continuation in ONE of the following three styles, choosing
whichever you think is best:
(1) Classical -- formal, literary, measured prose; elevated diction;
restrained, serious tone.
(2) Cute -- light, warm, endearing and wholesome; cozy details, gentle
humour, affectionate tone.
(3) Chuunibyou -- grandiose adolescent edginess; melodramatic
self-important narration, invented epithets and titles, dark
supernatural bombast.

Rules:
- Commit to a single style and sustain it throughout.
- Write at most 500 words.
- Output ONLY the continuation itself. Do not restate the prompt, do
not name the style you chose, and do not add any commentary, headings
or explanation.
```

The reward is produced by an LLM judge (`Qwen2.5-7B-Instruct`), which returns a style label and a quality verdict; the reward is the product of the two, i.e. credit requires both matching the hidden target style and passing the quality check.

Writing: judge system prompt

```
You are a strict but fair literary judge. You will be given ONE
passage of creative writing (a continuation of a story prompt).
Analyse it along exactly two dimensions.
```

```
## Dimension 1: STYLE
Decide which ONE of the following three styles best describes the
passage:
- Classical: formal, literary, measured prose; elevated or
old-fashioned diction; restrained, serious tone.
- Cute: light, warm, endearing, wholesome; small cozy details, gentle
humour, affectionate or childlike tone.
- Chuunibyou: grandiose adolescent edginess; melodramatic
self-important narration, invented epithets and titles, dark/flashy
supernatural bombast.
Choose the single closest match even if the fit is imperfect. Answer
with exactly one of: Classical, Cute, Chuunibyou.

## Dimension 2: QUALITY
Decide whether the passage is usable as a creative continuation.
Answer "good" if it is:
- recognisably relevant to a story prompt,
- understandable and coherent,
- reasonably well written,
- internally consistent enough to function as a continuation.
Answer "bad" ONLY when the passage is clearly unusable: incoherent or
nonsensical, off-topic, degenerate (looping or repeating
phrases/sentences with no purpose), empty or near-empty, not prose at
all (e.g. only meta-commentary or a list of instructions), or padded
with repeated filler purely to sound like a style.
You are NOT judging literary excellence. Ordinary, unremarkable but
readable writing is "good".
IMPORTANT: the passage was capped at 500 words and may have been cut
off by that limit. An abrupt ending, an unfinished final sentence, or
a story with no proper conclusion must NOT by itself make the quality
"bad". Judge only what is present.

## Output format
Reply with ONE line and nothing else, in exactly this format:
STYLE, QUALITY
For example:
Cute, good
Do not explain your reasoning.
```

Writing: judge user prompt

```
Judge the following passage.

<passage>
{response}
</passage>

Reply with one line: STYLE, QUALITY
```

B.3 SEARCH

Search: first step (no history)

```
You are an expert agent tasked with answering the given question
step-by-step.
Your question: {task_description}

Now it's your turn to respond for the current step. Choose only one of
the following actions (do not perform both):
(1) If you find you lack some knowledge, you can call a search engine
to get more external information using format: <search> your query
</search>.
```

```
(2) If you have enough knowledge to answer the question confidently,
provide your final answer within <answer> </answer> tags, without
detailed illustrations. For example, <answer>Beijing</answer>.
```

Search: later steps (with history)

```
You are an expert agent tasked with answering the given question
step-by-step.
Your question: {task_description}

Prior to this step, you have already taken {step_count} step(s). Below
is the interaction history where <search> </search> wrapped your past
search queries and <information> </information> wrapped the
corresponding search results returned by the external search engine.
History:
{memory_context}

Now it's your turn to respond for the current step. Choose only one of
the following actions (do not perform both):
(1) If you find you lack some knowledge, you can call a search engine
to get more external information using format: <search> your query
</search>.
(2) If you have enough knowledge to answer the question confidently,
provide your final answer within <answer> </answer> tags, without
detailed illustrations. For example, <answer>Beijing</answer>.
```

B.4 WEBSHOP

WebShop: first step (no history)

```
You are an expert autonomous agent operating in the WebShop e-commerce
environment.
Your task is to: {task_description}.
Your current observation is: {current_observation}.
Your admissible actions of the current situation are:
[
{available_actions}
].

Now it's your turn to take one action for the current step. Please
directly output the action.
```

WebShop: later steps (with history)

```
You are an expert autonomous agent operating in the WebShop e-commerce
environment.
Your task is to: {task_description}.
Prior to this step, you have already taken {step_count} step(s). Below
are the most recent {history_length} actions you took:
{action_history}
You are now at step {current_step} and your current observation is:
{current_observation}.
Your admissible actions of the current situation are:
[
{available_actions}
].

Now it's your turn to take one action for the current step. Please
directly output the action.
```

B.5 ALFWORLD

ALFWorld: first step (no history)

```
You are an expert agent operating in the ALFRED Embodied Environment.
Your current observation is: {current_observation}
Your admissible actions of the current situation are:
[{admissible_actions}].

Now it's your turn to take an action. Please directly output the
action.
```

ALFWorld: later steps (with history)

```
You are an expert agent operating in the ALFRED Embodied Environment.
Your task is to: {task_description}
Prior to this step, you have already taken {step_count} step(s). Below
are the most recent {history_length} observations and the
corresponding actions you took: {action_history}
You are now at step {current_step} and your current observation is:
{current_observation}
Your admissible actions of the current situation are:
[{admissible_actions}].

Now it's your turn to take an action. Please directly output the
action.
```

C ADDITIONAL RESULTS

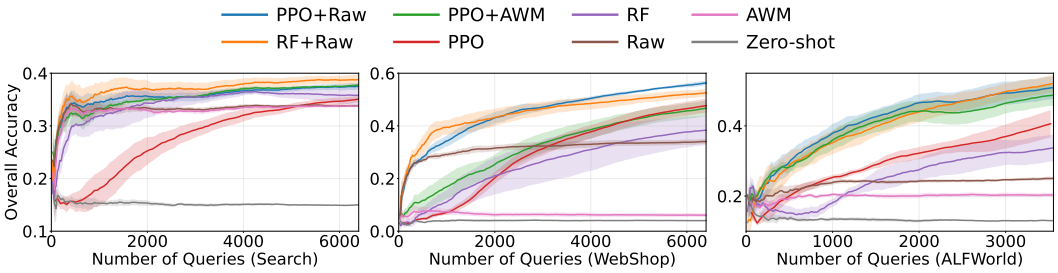


Figure 6: Overall accuracy on three agentic benchmarks: Search (left), WebShop (middle), and ALFWorld (right).