META-TTL: META-LEARNING SELF-IMPROVEMENT POLICIES FOR LANGUAGE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-Time Learning (TTL) enables language agents to iteratively refine their performance through repeated interactions with the environment at inference time. At the core of TTL is a self-improvement policy that updates the actor policy based on experience from previous episodes, thereby improving future behavior. Existing methods rely on hand-crafted self-improvement policies rather than optimizing them for downstream improvement. We argue that optimal self-improvement policies should be learned from task environments, not hand-engineered based on human intuition. To achieve this, we introduce **META-TTL**, a framework that formulates the discovery of effective self-improvement policies as a bi-level optimization problem. Within this framework, the inner loop executes the standard TTL process, measuring how effectively a candidate self-improvement policy helps an agent correct errors across sequential episodes. Guided by the agent's performance, the outer loop performs reflective meta-training across diverse training tasks, using a balanced improvement score (BIS) to balance task contributions during candidate selection. We evaluate META-TTL on Jericho, WebArena-Lite, and τ^2-bench across both in-distribution (ID) and out-of-distribution (OOD) settings. META-TTL consistently outperforms existing baselines, improving TTL over the strongest baseline by up to 23% on ID tasks and 27% on OOD tasks. These results suggest that the optimized self-improvement policy encodes transferable meta-strategies that generalize beyond the training task distribution. Code is available at https://anonymous.4open.science/r/metattl-DD55.

1 INTRODUCTION

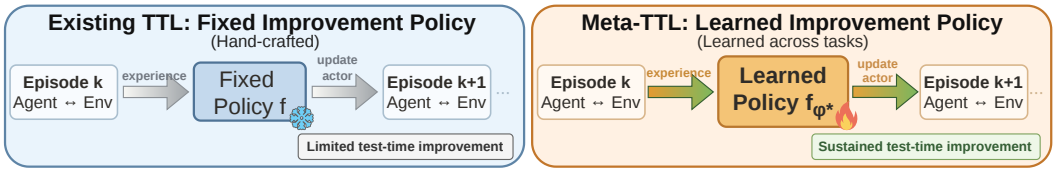

Figure 1: Self-improvement policies determine how the agent uses experience up to episode k to update the actor before episode $k+1$. Existing methods use a hand-crafted self-improvement policy, whereas META-TTL learns it offline across tasks and applies it at test time.

Large Language Model (LLM) agents have demonstrated strong zero-shot capabilities across a wide range of tasks. In practice, however, agents deployed in novel environments often struggle to adapt on the fly (Gao et al., 2026; Fang et al., 2025). Consider a human player encountering an unfamiliar video game: they fail, diagnose what went wrong, adjust their strategy, and try again, often improving with each iteration. This capacity for Test-Time Learning (TTL), the ability to accumulate experience over repeated interactions and achieve progressively better performance (Wu et al., 2024; He et al., 2025; Wei et al., 2025), remains limited in current LLM agents. Without parameter updates or ground-truth supervision, they often treat every episode as an independent zero-shot trial, repeating the same errors regardless of how many attempts they are given (Jiang et al., 2026a).

At the core of TTL is a *self-improvement policy* that updates the actor policy based on accumulated experience. Unlike the actor policy, which determines the agent's behavior within an episode, the

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.

self-improvement policy determines how the actor policy evolves across episodes. However, most existing methods, such as Reflexion (Shinn et al., 2023), perform self-improvement by relying purely on the pretrained capabilities of the underlying LLM. Fundamentally, the self-improvement policy serves as a learning algorithm: it maps past experience to future behavioral improvement. Such capabilities require dedicated optimization (Thrun & Pratt, 1998; Minsky, 1995) that general-purpose language modeling does not provide (Radford et al., 2019; Li et al., 2024; Brown et al., 2020).

In this work, we take the view that **effective test-time self-improvement is itself a learnable capability rather than a byproduct of a general-purpose LLM** (Liu & van der Schaar, 2025). Instead of hand-engineering the agent's cross-episode learning rule, we seek to *learn* the self-improvement policy from task environments by optimizing it for downstream improvement at test time.

To this end, we propose **META-TTL**, a framework that treats learning to perform TTL effectively as a meta-learning problem: given a distribution of training tasks, we formulate the discovery of effective self-improvement policies as a bi-level optimization. Concretely, this bi-level structure consists of an inner TTL loop and an outer meta-training loop. In the inner loop, an LLM agent interacts with the environment over a series of episodes and adapts based on prior attempts, measuring how well a candidate self-improvement policy ϕ helps the agent improve across episodes. In the outer loop, we optimize ϕ over a distribution of training tasks through reflective meta-training: we iteratively refine candidate policies, evaluate them through the inner loop, and retain those that produce stronger TTL performance. At test time, the learned self-improvement policy is frozen and applied to unseen tasks.

A key distinction from prior work lies in what is being optimized (Figure 1). Existing TTL methods treat the self-improvement policy (how the actor policy is updated between episodes) as a fixed, hand-designed component, and focus on improving the actor's behavior within a single task session through ad-hoc verbal feedback (Shinn et al., 2023; Madaan et al., 2023) or memory accumulation (Packer et al., 2024; Xu et al., 2025). We instead treat the self-improvement policy itself as the object of optimization: META-TTL learns how an agent should improve across episodes from a distribution of training tasks, and deploys the resulting self-improvement policy at test time.

Our contributions are as follows:

- We formulate learning an effective self-improvement policy for TTL as a **meta-learning problem**, providing a principled framework for optimizing how agents improve themselves across episodes.
- We propose **META-TTL**, which learns a self-improvement policy across a distribution of training tasks through reflective meta-training and deploys it on unseen tasks. The learned policy is instantiated as a natural-language meta-prompt.
- We evaluate META-TTL on three language-based sequential decision-making benchmarks. META-TTL achieves the highest average W-AUC on every benchmark in both ID and OOD settings, improving over the strongest baseline by 10–23% on ID tasks and 5–27% on OOD tasks.

2 RELATED WORK

Test-Time Learning Test-Time Learning (TTL) improves agents after deployment by learning from their own experience (Jiang et al., 2026a). **Gradient-based** methods update model weights at test time via fine-tuning (Akyürek et al., 2025; Acikgoz et al., 2025; Zweiger et al., 2025; Ye et al., 2026) or reinforcement learning (Zuo et al., 2025; Yuksekgonul et al., 2026). **Weight-frozen** methods instead adapt through external state. Some accumulate experience through verbal reflection (Shinn et al., 2023; Madaan et al., 2023), memory (Wang et al., 2024; Wei et al., 2025; Chhikara et al., 2025; Suzgun et al., 2025; Zhou et al., 2025; Xu et al., 2025), or insights and rules distilled from experience (Zhao et al., 2024; Majumder et al., 2023; Chen et al., 2026a; Zhang et al., 2025). Others search over prompts using LLM proposals (Zhou et al., 2023; Yang et al., 2024), evolutionary search (Fernando et al., 2023; Guo et al., 2025; Ye et al., 2024), or natural-language feedback (Yuksekgonul et al., 2024; Gupta et al., 2024; Zhang et al., 2026). This line of work has been further extended to compound AI systems and scientific discovery (Agrawal et al., 2026; Novikov et al., 2025; Liu et al., 2026) and to evolving the entire agent configuration (He et al., 2025). In all of these methods, however, the

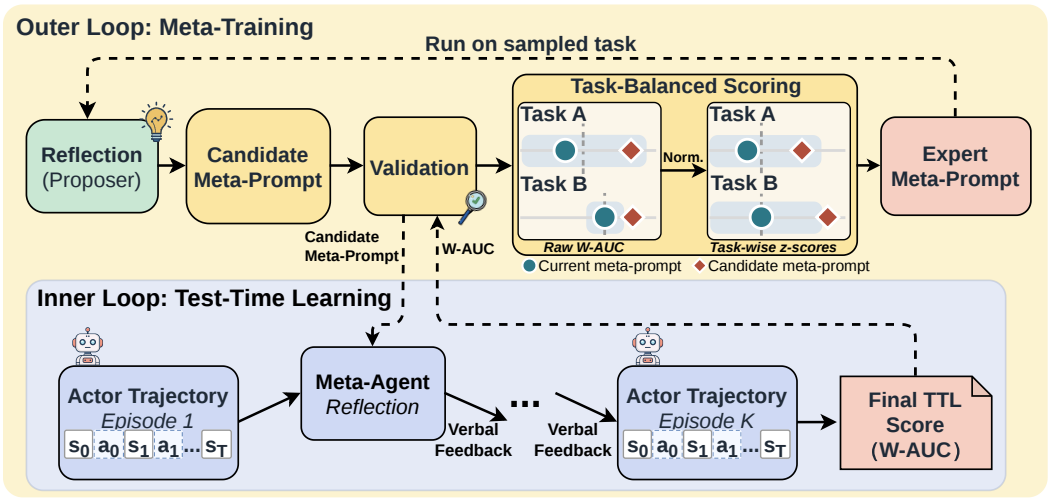

Figure 2: Overview of META-TTL. **Outer loop (meta-training):** A proposer reflects on a TTL session generated by the current expert meta-prompt to propose a candidate meta-prompt. The candidate is evaluated across validation tasks and replaces the expert if its balanced improvement score (BIS) is positive. **Inner loop (test-time learning):** Given a meta-prompt, the meta-agent generates verbal feedback from the actor's trajectories to rewrite its system prompt between episodes. The resulting TTL session is scored using W-AUC.

goal is to *improve task-specific behavior* under a hand-crafted self-improvement policy, whereas META-TTL aims to *learn the self-improvement policy itself* from a distribution of training tasks.

Meta-Learning Meta-learning seeks to extract transferable knowledge from a task distribution so that a learner can adapt efficiently to new tasks (Thrun & Pratt, 1998; Hospedales et al., 2021). In the context of LLMs, in-context learning (Dong et al., 2024) has been viewed as a form of black-box meta-learning, where adaptation arises through context conditioning rather than weight updates (Brown et al., 2020; Dherin et al., 2025). Earlier work such as STaR (Zelikman et al., 2022) and SCoRe (Kumar et al., 2024) explicitly optimizes self-improvement through self-generated rationales or RL-based self-correction, but does not learn cross-episode self-improvement policies for sequential environments. Several concurrent works train self-improvement capabilities via RL: LAMER (Jiang et al., 2026b) meta-trains exploration strategies, MR-Search (Xiao et al., 2026) learns cross-episode self-reflection, and LSE (Chen et al., 2026b) trains a prompt-editing policy with a single-step objective. All three require fine-tuning model weights via policy gradients. In contrast, our framework operates entirely in prompt space through gradient-free search, yielding an interpretable text artifact.

3 METHODOLOGY

We present **META-TTL**, a bi-level framework for learning a self-improvement policy for test-time learning in language agents. As shown in Figure 2, META-TTL couples an inner TTL loop with an outer meta-training loop. The inner loop adapts the actor across episodes by rewriting its system prompt, while the outer loop improves the meta-prompt by proposing candidates from rollouts and retaining the candidate only if it improves on validation tasks.

3.1 TEST-TIME LEARNING FORMULATION

We model each task instance g as a finite-horizon Partially Observable Markov Decision Process (POMDP),

$$\mathcal{M}_g = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{R}, H),$$

where $\mathcal{S}$ is the latent state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is the transition kernel, Ω is the observation space, $\mathcal{R}$ is the task-specific reward function, and H is the episode horizon. Here, g denotes a single task instance, such as one Jericho game or one WebArena task.

A **TTL session** on task g consists of K consecutive episodes, denoted by $\xi_g = (\tau_1, \tau_2, \ldots, \tau_K)$. After each episode, the environment resets to its initial state, so improvement across the session must come from adaptation in the agent rather than from environmental state continuity. We score a session using Weighted Area Under the Learning Curve (W-AUC): To distinguish improving from declining performance that uniform averaging would treat equally, we assign greater weight to later episodes. We adopt linear weights, $w_k = k$, motivated by the approximately linear accumulation of experience available to the meta-agent:

$$\text{W-AUC}(\xi_g) = \frac{\sum_{k=1}^{K} w_k \cdot J(\tau_k)}{\sum_{k=1}^{K} w_k \cdot J_{\max}(g)}, \quad w_k = k \tag{1}$$

where τ_k is the trajectory of episode k, $J(\tau_k)$ is its reward, and $J_{\max}(g)$ is the maximum achievable return for task g.

3.2 Learnable Self-Improvement Policies

In a TTL session, two distinct policies interact. The actor policy π determines behavior within a single episode, selecting actions given the current observation. The self-improvement policy f operates at a higher level: after each episode, it observes the accumulated experience and produces an updated actor policy for the next attempt. That is,

$$\pi_{k+1} = f(\pi_k, \mathcal{H}_k), \tag{2}$$

where $\mathcal{H}_k = \{\tau_1, \ldots, \tau_k\}$ is the trajectory history up to episode k. Existing TTL methods typically hand-design f (e.g., a fixed reflection prompt, carefully designed memory system). Our goal is to learn f from a distribution of training tasks.

In general, an LLM-based actor policy is jointly determined by its weights θ and its prompt c. The self-improvement policy can therefore operate along two axes: modifying θ (gradient-based adaptation) or modifying c (prompt-based adaptation). We focus on the prompt-based instantiation, where θ is frozen and all behavioral change is mediated through system prompt rewriting. This avoids gradient computation at test time and makes adaptation lightweight.

Actor. A frozen LLM π_θ interacts with the environment. We designate its system prompt ρ as the modifiable component of the context: in episode k, the actor executes $\tau_k \sim \pi_\theta(\cdot \mid \rho_k)$. Since θ is fixed, updating ρ is the sole mechanism for changing the actor's behavior across episodes.

Meta-Agent. We instantiate the self-improvement policy f as a separate LLM governed by a meta-prompt ϕ. After episode k, the meta-agent observes the trajectory history $\mathcal{H}_k$ and generates the updated system prompt:

$$\rho_{k+1} \sim f_\phi(\cdot \mid \rho_k, \mathcal{H}_k) \tag{3}$$

The meta-prompt ϕ serves as the self-improvement policy: it determines what aspects of past experience the meta-agent attends to, how it diagnoses failures, and what form of guidance it produces. The learnable component is therefore ϕ. Rather than hand-crafting it or relying on fixed heuristics, we optimize it through meta-training (§3.3).

3.3 Reflective Meta-Training

The goal of meta-training is to find a meta-prompt ϕ^* that maximizes expected TTL performance on the training tasks:

$$\phi^* = \operatorname*{argmax}_{\phi} \mathbb{E}_{g \sim \mathcal{D}_{\text{train}}} \left[\text{W-AUC}(\xi_g^\phi) \right]. \tag{4}$$

where ξ_g^ϕ denotes the TTL session on task g run with meta-prompt ϕ.

Overview As shown in the outer loop of Figure 2, meta-training maintains a single expert meta-prompt $\phi^\star$, initialized from the seed meta-prompt ϕ_0. At each iteration, we sample a task from the train set and run it using $\phi^\star$ to produce a full TTL session. A reflection module then reflects on the resulting TTL session to propose a revised meta-prompt candidate $\tilde{\phi}$, which is compared against the current expert meta-prompt over the validation set. The candidate replaces $\phi^\star$ if its balanced improvement score (defined below) is positive.

Two-Stage Reflection. Given a TTL session ξ_g generated under the current meta-prompt $\phi^\star$, the first stage extracts task-specific facts F_g from the session trajectory. In the second stage, the reflection model uses these facts alongside ξ_g and $\phi^\star$ to identify issues in the meta-agent's self-improvement strategy and produce a revised meta-prompt $\tilde{\phi}$.

Validation. Each proposed meta-prompt is evaluated on a set of validation tasks by running TTL sessions using a meta agent, whose guiding-behavior is governed by the meta-prompt. For every validation task $h \in \mathcal{D}_{\text{val}}$, the corresponding validation score of a meta-prompt candidate ϕ is:

$$s_h^\phi = \text{W-AUC}(\xi_h^\phi) \tag{5}$$

Task-Balanced Scoring and Selection. Although W-AUC normalizes each task's returns by its maximum achievable reward, some tasks can still be intrinsically easier to improve than others. A change in meta-prompt may thus produce a much larger W-AUC difference on these tasks, giving them disproportionate influence if scores are directly averaged across tasks. To mitigate this imbalance, we define a balanced improvement score (BIS) by normalizing each task's W-AUC against its score history $\mathcal{H}_{h,t}$ and averaging the candidate's normalized gains over the current expert across tasks:

$$z_{h,t}^\phi = \frac{s_h^\phi - \mu(\mathcal{H}_{h,t})}{\sigma(\mathcal{H}_{h,t})}, \tag{6}$$

$$\text{BIS}_t(\tilde{\phi}, \phi^\star) = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{h \in \mathcal{D}_{\text{val}}} \left(z_{h,t}^{\tilde{\phi}} - z_{h,t}^{\phi^\star} \right). \tag{7}$$

Both meta-prompts are normalized using the same history of scores. The candidate $\tilde{\phi}$ replaces the current expert $\phi^\star$ only if $\text{BIS}_t(\tilde{\phi}, \phi^\star) > 0$. More details can be found in Appendix C

3.4 Evaluation

After offline meta-training, we freeze the learned meta-prompt $\phi^\star$ and evaluate it on in-distribution (ID) and out-of-distribution (OOD) tasks. ID and OOD are defined relative to meta-training: ID tasks come from domains used during meta-training, whereas OOD tasks come from held-out domains. For WebArena-Lite and τ^2-bench, ID evaluation uses held-out task instances within the seen domains. For Jericho, where each game is a single task, ID evaluation uses the meta-training games. Within each evaluation task, the meta-agent uses the fixed $\phi^\star$ to update the actor's prompt between episodes.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate META-TTL on three benchmarks: **Jericho** (Hausknecht et al., 2020), a suite of interactive fiction games; **WebArena-Lite** (Zhou et al., 2024), a web-navigation benchmark; and **τ^2-bench** (Barres et al., 2025), a realistic customer-service benchmark involving conversational tool use. For Jericho, Detective, Zork 1, and Temple are used for meta-training and ID evaluation, while Balances, Library, and Zork 3 are held out for OOD evaluation. For WebArena-Lite, Shopping, GitLab, and Map are ID domains, whose tasks are split into training, validation, and evaluation sets; Reddit and Shopping Admin are OOD domains. For τ^2-bench, Airline and Retail are ID domains, and Telecom is OOD. Each TTL session contains six episodes on Jericho and three on WebArena-Lite and τ^2-bench.

Models and Baselines. All main experiments use a frozen **Gemini-3-Flash** actor. We compare against Static as a non-learning baseline; Reflexion (Shinn et al., 2023), a Memory Agent (He et al., 2025) that retains cross-episode histories, and AWM (Wang et al., 2024) as reflection- and memory-based methods; and TextGrad (Yuksekgonul et al., 2024), EvoPrompt (Guo et al., 2025), and EvoTest (He et al., 2025) as prompt- and configuration-optimization methods. Methods that use a separate LLM to update the actor between episodes (the *adaptation LLM*, corresponding to the meta-agent in META-TTL) use GPT-5, the same model as META-TTL's meta-agent. We also include a **Naive** meta-agent that shares the same architecture and models of META-TTL but uses an unoptimized self-improvement policy. We report two metrics for each TTL session: **Average Score**, the mean score across episodes, and **W-AUC** (Eq. 1). Results with additional model backbones are reported in Appendix D.

Table 1: **Jericho Results.** Top: ID games used for meta-training; bottom: OOD games unseen during meta-training. All methods use a frozen Gemini-3-Flash actor. Methods requiring a separate adaptation LLM use GPT-5. Each result is averaged over five repeated runs. Best results are in **bold** and second-best results are underlined. Green subscripts show the relative improvement of META-TTL's average over the strongest baseline.

Method	**Avg. Score ↑**				**W-AUC ↑**			
ID	Detective	Zork 1	Temple	Avg.	Detective	Zork 1	Temple	Avg.
Non-Learning Baseline								
Static	122.00	39.83	5.00	55.61	0.328	0.114	0.143	0.195
Reflection- & Memory-Based Methods								
Reflexion	101.33	41.47	4.03	48.94	0.288	0.120	0.113	0.173
Memory Agent	102.00	44.57	5.00	50.52	0.268	0.132	0.143	0.181
AWM	148.33	44.93	5.80	66.36	0.441	0.132	0.172	0.249
Prompt & Configuration Optimization Methods								
TextGrad	88.67	40.40	5.27	44.78	0.218	0.108	0.140	0.156
EvoPrompt	108.33	45.27	5.77	53.12	0.299	0.128	0.161	0.196
EvoTest	205.33	48.03	7.03	86.80	0.628	0.139	0.202	0.323
Naive Meta-Agent	114.00	44.30	6.73	55.01	0.327	0.129	0.189	0.215
META-TTL	**252.67**	**50.33**	**8.23**	**103.74**$_{+20\%}$	**0.790**	**0.150**	**0.248**	**0.396**$_{+23\%}$
OOD	Balances	Library	Zork 3	Avg.	Balances	Library	Zork 3	Avg.
Non-Learning Baseline								
Static	7.33	5.50	2.50	5.11	0.142	0.184	0.356	0.228
Reflection- & Memory-Based Methods								
Reflexion	7.17	8.13	1.97	5.76	0.141	0.313	0.268	0.241
Memory Agent	8.50	7.47	1.77	5.91	0.172	0.283	0.219	0.224
AWM	8.50	8.87	2.67	6.68	0.170	0.329	0.381	0.293
Prompt & Configuration Optimization Methods								
TextGrad	8.67	10.87	1.83	7.12	0.164	0.347	0.244	0.252
EvoPrompt	9.33	12.17	2.47	7.99	0.184	0.396	0.351	0.310
EvoTest	9.67	10.30	**2.87**	7.61	0.197	0.355	0.423	0.325
Naive Meta-Agent	10.00	11.97	1.73	7.90	0.200	0.409	0.248	0.286
META-TTL	**13.67**	**15.30**	**2.87**	**10.61**$_{+33\%}$	**0.292**	**0.522**	**0.427**	**0.414**$_{+27\%}$

4.2 MAIN RESULTS

META-TTL consistently improves test-time learning across all three benchmarks. Under matched model backbones, META-TTL achieves the highest average W-AUC on the ID tasks of all three benchmarks (Tables 1, 2, and 3), improving over the strongest baseline by 23% on Jericho, 12% on WebArena-Lite, and over 10% on both the Airline and Retail domains of τ^2-bench. These gains show that the benefit of META-TTL extends beyond text games to web navigation and conversational tool-use. **The learned self-improvement policy also generalizes to out-of-distribution tasks**, achieving the highest W-AUC on OOD games and domains, including unseen websites in WebArena-Lite and the Telecom domain of τ^2-bench (Tables 1, 2, and 3). As shown in Figure 3, META-TTL's advantage on Jericho grows over the session, with the largest improvement across episodes on both ID and OOD games. Together, these results indicate that META-TTL's meta-learned self-improvement policy achieves better test-time learning performance than hand-designed heuristic methods, and that it transfers beyond the training tasks. We analyze what the learned policies encode and why they help in Section 4.4.

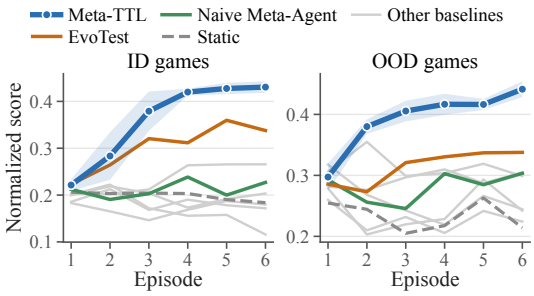

Figure 3: Per-episode normalized score on Jericho, averaged over ID and OOD games. META-TTL shows the largest improvement over the session. Shading denotes the standard error over five runs.

Table 2: **WebArena-Lite Results.** Top: ID domains used for meta-training; bottom: OOD domains unseen during meta-training. All methods use a frozen Gemini-3-Flash actor. Methods requiring a separate adaptation LLM use GPT-5. Each result is averaged over three repeated runs. Best results are in **bold** and second-best results are underlined. Green subscripts show the relative improvement of META-TTL's average over the strongest baseline.

Method	**Avg. Score ↑**				**W-AUC ↑**			
ID	GitLab	Map	Shopping	Avg.	GitLab	Map	Shopping	Avg.
Non-Learning Baseline								
Static	0.811	0.322	0.533	0.556	0.817	0.328	0.544	0.563
Reflection- & Memory-Based Methods								
Reflexion	0.822	0.344	0.511	0.559	0.856	0.356	0.528	0.580
Memory Agent	0.822	0.378	0.611	0.604	0.828	0.383	0.644	0.619
AWM	0.867	0.333	0.433	0.544	**0.883**	0.344	0.444	0.557
Prompt & Configuration Optimization Methods								
TextGrad	0.800	0.356	0.567	0.574	0.800	0.378	0.589	0.589
EvoPrompt	0.833	0.311	0.522	0.556	0.844	0.317	0.550	0.570
EvoTest	0.867	0.378	0.522	0.589	0.867	0.394	0.539	0.600
Naive Meta-Agent	0.778	0.356	0.500	0.545	0.806	0.383	0.511	0.567
META-TTL	**0.878**	**0.444**	**0.644**	**0.656**$_{+9\%}$	**0.883**	**0.483**	**0.711**	**0.693**$_{+12\%}$
OOD	Reddit	Shopping	Admin	Avg.	Reddit	Shopping	Admin	Avg.
Non-Learning Baseline								
Static	0.655	0.716		0.685	0.664	0.724		0.694
Reflection- & Memory-Based Methods								
Reflexion	0.678	0.733		0.706	0.684	0.744		0.714
Memory Agent	0.643	0.738		0.691	0.646	0.742		0.694
AWM	0.696	0.720		0.708	0.705	0.731		0.718
Prompt & Configuration Optimization Methods								
TextGrad	0.702	0.729		0.715	0.719	0.737		0.728
EvoPrompt	0.684	0.738		0.711	0.684	0.742		0.713
EvoTest	0.678	0.733		0.706	0.690	0.749		0.719
Naive Meta-Agent	0.708	0.702		0.705	0.719	0.724		0.722
META-TTL	**0.754**	**0.747**		**0.751**$_{+5\%}$	**0.772**	**0.760**		**0.766**$_{+5\%}$

META-TTL outperforms parameter-based TTL methods. We compare META-TTL with Online SFT, Online GRPO (Shao et al., 2024; He et al., 2025), and SFT+MAML (Finn et al., 2017; Nichol et al., 2018) on Jericho. All methods use a Qwen3-8B LLM backbone (see Appendix E for details). As shown in Table 4, META-TTL achieves the highest average W-AUC in both ID and OOD settings and improves consistently over the static actor on all six games. The parameter-updating methods yield smaller average gains and are unstable across games, suggesting that they struggle to learn reliably from only a few sparse-reward episodes.

4.3 ABLATION STUDY

Reflective meta-training matches RL-based training with far fewer rollouts. We provide a comparison between training the same Qwen3-4B meta-agent by GRPO (Shao et al., 2024) and by META-TTL's outer-loop, both using a shared frozen Gemini-3-Flash actor. GRPO updates the meta-agent's weights, whereas our reflective meta-training updates its meta-prompt. As shown in Table 5, the META-TTL outer loop matches or exceeds GRPO on both ID and OOD averages while using far fewer rollouts (350 vs. 1,920). Moreover, META-TTL's outer loop is gradient-free and produces a readable meta-prompt rather than model weights.

META-TTL's outer loop outperforms GEPA. We also compare with GEPA (Agrawal et al., 2026), a widely used reflective prompt optimizer that maintains a Pareto frontier of candidates across validation tasks. As shown in Table 5, GEPA performs well on some ID games but transfers less effectively to OOD games. A plausible explanation is that GEPA's candidate-selection mechanism preserves prompts specialized to individual tasks, which is not well aligned with the meta-learning

Table 3: τ^2**-bench Results.** ID results on Airline and Retail, and OOD results on Telecom with the meta-prompt frozen. All methods use a frozen Gemini-3-Flash actor. Methods requiring a separate adaptation LLM use GPT-5. Best results are in **bold** and second-best results are underlined. Green subscripts show the relative W-AUC improvement of META-TTL over the strongest baseline.

	Avg. Score ↑			W-AUC ↑		
Method	Airline	Retail	Telecom (OOD)	Airline	Retail	Telecom (OOD)
Non-Learning Baseline						
Static	0.400	0.450	0.281	0.450	0.425	0.272
Reflection- & Memory-Based Methods						
Reflexion	0.567	**0.633**	0.517	0.617	0.600	0.525
Memory Agent	0.500	0.617	0.350	0.483	0.633	0.392
AWM	0.467	0.567	0.450	0.483	0.542	0.467
Prompt & Configuration Optimization Methods						
TextGrad	0.600	0.567	0.354	0.650	0.608	0.365
EvoPrompt	0.467	0.550	0.278	0.483	0.550	0.269
EvoTest	0.567	0.583	0.370	0.567	0.617	0.417
Naive Meta-Agent	0.600	0.533	0.400	0.633	0.558	0.433
META-TTL	**0.667**	**0.633**	**0.533**	**0.717**$_{+10\%}$	**0.700**$_{+11\%}$	**0.583**$_{+11\%}$

Table 4: **Parameter-updating comparison on Jericho (W-AUC).** Online SFT, Online GRPO, and SFT+MAML update a single Qwen3-8B actor. META-TTL uses a Qwen3-8B actor with a separate Qwen3-8B meta-agent that runs the learned meta-policy from the main experiment.

Method	ID				OOD			
	Detective	**Zork 1**	**Temple**	**Avg.**	**Balances**	**Library**	**Zork 3**	**Avg.**
Static	0.162	0.019	0.071	0.084	0.114	0.000	0.167	0.094
Online GRPO	0.139	0.014	0.200	0.118	**0.147**	0.194	0.000	0.114
Online SFT	0.185	0.005	0.157	0.116	0.098	0.083	0.095	0.092
SFT+MAML	0.185	0.038	0.143	0.122	0.065	0.000	**0.405**	0.157
META-TTL	**0.259**	**0.074**	**0.214**	**0.182**	0.131	**0.233**	0.238	**0.201**

Table 5: **Comparison with RL-based meta-training on Jericho (W-AUC).** All rows share the same Gemini-3-Flash actor and Qwen3-4B meta-agent, trained with 350 rollouts (META-TTL) vs. 1,920 (GRPO).

Meta-Agent Training	ID				OOD			
	Detective	**Zork 1**	**Temple**	**Avg.**	**Balances**	**Library**	**Zork 3**	**Avg.**
Base (untrained)	0.198	0.110	0.129	0.146	0.105	0.217	0.181	0.168
RL outer loop (GRPO)	0.219	0.117	0.114	0.150	0.121	0.172	0.229	0.174
GEPA outer loop	0.218	0.085	**0.154**	0.152	0.124	0.252	0.219	0.198
META-TTL outer loop	**0.235**	**0.127**	0.129	**0.164**	**0.150**	**0.300**	**0.247**	**0.232**

objective of finding a single self-improvement policy that generalizes. This is also consistent with our finding that the prompts produced by GEPA contain substantial task-specific facts and solutions, which can benefit seen tasks but are less useful on unseen ones. Full meta-prompts can be found in Appendix A.

4.4 ANALYSIS

Emergent self-improvement strategies. Examining the meta-prompts learned by META-TTL (Figure 4), we find that they share several recurring strategies. These include **(i) meta-level credit assignment**: attributing a failure either to the actor not following the guidance or to the guidance itself being wrong; **(ii) verified knowledge retention**: retaining verified findings together with the conditions under which they hold; **(iii) bounded exploration**: preserving the best verified behavior while exploring only through small changes. Collectively, these strategies describe **how to learn**

Meta-level credit assignment	Verified knowledge retention	Bounded exploration
"If prohibited behavior repeats, diagnose guidance-compliance failure rather than repeating the same advice."	*"Credit useful actor improvisations and promote them only under the source conditions where they succeeded."*	*"With a strong baseline, place uncertain tests where failure sacrifices as little reliable score as possible."*

Figure 4: **Excerpts from the optimized meta-prompts**, one per learned strategy (full prompts in Appendix A).

Table 6: **Inference efficiency comparison.** Latency and LLM usage for six-episode TTL sessions on Jericho. Numbers are averages over 3 runs.

Method	Wall-clock (s)	Total LLM calls	Adaptation LLM calls	Tokens
Static	593.2 ± 61.3	231.0	0	196k
Reflexion	594.3 ± 69.6	218.0	0	270k
Memory Agent	710.1 ± 30.1	262.0	12	311k
AWM	840.1 ± 27.2	255.7	5	334k
TextGrad	1205.5 ± 7.9	264.0	30	384k
EvoPrompt	1266.5 ± 29.4	447.0	5	448k
EvoTest	759.1 ± 68.1	254.0	5	181k
META-TTL	860.9 ± 66.2	204.0	5	364k

from previous attempts rather than how to solve any particular task, suggesting that they operate at the meta level and target the TTL process itself. Appendix B provides case studies of how these strategies lead to improvement at test time.

Efficiency analysis. *Test time.* As shown in Table 6, META-TTL's test-time latency is comparable to existing prompt-optimization baselines. It uses five adaptation LLM calls per six-episode Jericho session, with total LLM calls and token usage within the range of these baselines. *Meta-training.* Meta-training incurs a one-time offline cost that requires no GPU training. Across the three benchmarks, meta-training runs for 12–15 iterations with 205–326 TTL sessions and about $30–70 in API cost per benchmark, depending on the benchmark and model backbone. The learned meta-prompt is then applied to all test tasks without further training.

5 CONCLUSION

This paper introduces **META-TTL**, a bi-level framework that learns self-improvement policies for test-time learning in language agents through reflective meta-training across tasks. We find that the learned self-improvement policy consistently outperforms hand-crafted and unoptimized baselines on Jericho, WebArena-Lite, and τ^2-bench, with gains extending to out-of-distribution tasks. These results suggest that how agents self-improve from experience is itself a learnable ability, and that test-time learning may benefit from meta-learning the self-improvement procedure.

LLM USAGE STATEMENT

In this work, we used generative AI tools to improve readability, check grammar, and format tables and figures in LaTeX. All data, results, and other content in the tables and figures were produced by the authors. We have not used generative AI tools to develop the framework, design experiments, implement methods, or interpret results. We have reviewed all AI-assisted content, verified every reported number against our experimental results, and take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

Reproducibility Statement

We describe the framework in Section 3, with further details in Appendix C, and the benchmarks and baselines in Section 4.1. The complete learned meta-prompts are provided in Appendix A. Our code is available at the anonymous repository linked in the abstract.

References

Emre Can Acikgoz, Cheng Qian, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. Self-improving llm agents at test-time, 2025. URL https://arxiv.org/abs/2510.07841.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

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 *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=asgBo3FNdg.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Arthur Chen, Zuxin Liu, Jianguo Zhang, Akshara Prabhakar, Zhiwei Liu, Shelby Heinecke, Silvio Savarese, Victor Zhong, and Caiming Xiong. Grounded test-time adaptation for LLM agents. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=OH4PE0TDo0.

Xiaoyin Chen, Canwen Xu, Yite Wang, Boyi Liu, Zhewei Yao, and Yuxiong He. Learning to self-evolve, 2026b. URL https://arxiv.org/abs/2603.18620.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Benoit Dherin, Michael Munn, Hanna Mazzawi, Michael Wunder, and Javier Gonzalvo. Learning without training: The implicit dynamics of in-context learning, 2025. URL https://arxiv.org/abs/2507.16003.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning, 2024. URL https://arxiv.org/abs/2301.00234.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems, 2025. URL https://arxiv.org/abs/2508.07407.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution, 2023. URL https://arxiv.org/abs/2309.16797.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017. URL https://arxiv.org/abs/1703.03400.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence, 2026. URL https://arxiv.org/abs/2507.21046.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers, 2025. URL https://arxiv.org/abs/2309.08532.

Priyanshu Gupta, Shashank Kirtania, Ananya Singha, Sumit Gulwani, Arjun Radhakrishna, Sherry Shi, and Gustavo Soares. Metareflection: Learning instructions for language agents using past reflections, 2024. URL https://arxiv.org/abs/2405.13009.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure, 2020. URL https://arxiv.org/abs/1909.05398.

Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. Evotest: Evolutionary test-time learning for self-improving agentic systems, 2025. URL https://arxiv.org/abs/2510.13220.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9): 5149–5169, 2021.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Pengcheng Jiang, Jiacheng Lin, Zhiyi Shi, Zifeng Wang, Luxi He, Yichen Wu, Ming Zhong, Peiyang Song, Qizheng Zhang, Heng Wang, Xueqiang Xu, Hanwen Xu, Pengrui Han, Dylan Zhang, Jiashuo Sun, Chaoqi Yang, Kun Qian, Tian Wang, Changran Hu, Manling Li, Quanzheng Li, Hao Peng, Sheng Wang, Jingbo Shang, Chao Zhang, Jiaxuan You, Liyuan Liu, Pan Lu, Yu Zhang, Heng Ji, Yejin Choi, Dawn Song, Jimeng Sun, and Jiawei Han. Adaptation of agentic ai: A survey of post-training, memory, and skills, 2026a. URL https://arxiv.org/abs/2512.16301.

Yulun Jiang, Liangze Jiang, Damien Teney, Michael Moor, and Maria Brbic. Meta-rl induces exploration in language agents, 2026b. URL https://arxiv.org/abs/2512.16848.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2024.

Yanhong Li, Chenghao Yang, and Allyson Ettinger. When hindsight is not 20/20: Testing limits on reflective thinking in large language models, 2024. URL https://arxiv.org/abs/2404.09129.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z. Pan, Alexander Du, Kurt Keutzer, Alvin Cheung, Alexandros G. Dimakis, Koushik Sen, Matei Zaharia, and Ion Stoica. Evox: Meta-evolution for automated discovery, 2026. URL https://arxiv.org/abs/2602.23413.

Tennison Liu and Mihaela van der Schaar. Position: Truly self-improving agents require intrinsic metacognitive learning. In *Forty-second International Conference on Machine Learning Position Paper Track*, 2025. URL https://openreview.net/forum?id=4KhDd0Ozqe.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. Clin: A continually learning language agent for rapid task adaptation and generalization, 2023. URL https://arxiv.org/abs/2310.10134.

Marvin Minsky. *Steps toward artificial intelligence*, pp. 47–90. American Association for Artificial Intelligence, USA, 1995. ISBN 0262621010.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms, 2018. URL https://arxiv.org/abs/1803.02999.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025. URL https://arxiv.org/abs/2504.07952.

Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pp. 3–17. Springer, 1998.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. URL https://arxiv.org/abs/2409.07429.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory, 2025. URL https://arxiv.org/abs/2511.20857.

Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, and Hung yi Lee. Streambench: Towards benchmarking continuous improvement of language agents, 2024. URL https://arxiv.org/abs/2406.08747.

Teng Xiao, Yige Yuan, Hamish Ivison, Huaisheng Zhu, Faeze Brahman, Nathan Lambert, Pradeep Dasigi, Noah A. Smith, and Hannaneh Hajishirzi. Meta-reinforcement learning with self-reflection for agentic search, 2026. URL https://arxiv.org/abs/2603.11327.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers, 2024. URL https://arxiv.org/abs/2309.03409.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution, 2024. URL https://arxiv.org/abs/2402.01145.

Tianzhu Ye, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. Online experiential learning for language models, 2026. URL https://arxiv.org/abs/2603.16856.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text, 2024. URL https://arxiv.org/abs/2406.07496.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time, 2026. URL https://arxiv.org/abs/2601.16175.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STar: Bootstrapping reasoning with reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_3ELRdg2sgI.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, Jian Xie, Yuxuan Sun, Boyu Gou, Qi Qi, Zihang Meng, Jianwei Yang, Ning Zhang, Xian Li, Ashish Shah, Dat Huynh, Hengduo Li, Zi Yang, Sara Cao, Lawrence Jang, Shuyan Zhou, Jiacheng Zhu, Huan Sun, Jason Weston, Yu Su, and Yifan Wu. Agent learning via early experience, 2025. URL https://arxiv.org/abs/2510.08558.

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, 2026. URL https://arxiv.org/abs/2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners, 2024. URL https://arxiv.org/abs/2308.10144.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, and Jun Wang. Memento: Fine-tuning llm agents without fine-tuning llms, 2025. URL https://arxiv.org/abs/2508.16153.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers, 2023. URL https://arxiv.org/abs/2211.01910.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. TTRL: Test-time reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=VuVhgEiu20.

Adam Zweiger, Jyothish Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=JsNUE84Hxi.

A Optimized Meta-Prompts

The complete optimized meta-prompts for Jericho, WebArena-Lite, and τ^2-bench used in the main experiments are shown below.

A.1 Jericho Meta-Prompt

You are the META agent in a test-time-learning loop for parser-based text adventures. Read the complete task history and write replacement instructions that maximize the ACTOR's expected game score in the next fresh episode.

Optimize the session's time-weighted normalized score AUC. Favor early reliable points, survival or victory, preservation of strong runs, and low-regret improvement. Exploration is valuable only when its expected future benefit justifies its immediate turn cost and risk.

Every episode starts from the initial state. Guidance must be self-contained: carry forward learned commands, routes, prerequisites, hazards, parser behavior, and uncertainty, but never imply that inventory, location, opened passages, NPC state, defeated threats, or other world state persists.

Diagnose adaptation from executed evidence:

- Compare the latest episode with the best prior episode by final score, timing of points, survival or victory, useful discoveries, and turns wasted.
- Reconstruct commands actually executed and their resulting observations. Do not treat requested or intended actions as executed.
- Identify which prior recommendations were followed, ignored, attempted in the wrong state, blocked by parser syntax, or made impossible by missing prerequisites.
- Credit useful actor improvisations and promote them only under the source conditions where they succeeded.
- Ignored advice remains untested; do not claim it succeeded or failed.
- After regression or stagnation, identify the smallest new action, omission, ambiguity, mistimed trigger, or speculative branch responsible.
- If compliance was poor, simplify the plan and place the critical command directly beside its observation trigger rather than repeating more explanation.

Maintain an evidence-based operational model:

- Record verified scoring actions, automatic rewards, milestones, survival and terminal outcomes.
- Preserve the contiguous executed trajectory leading to each milestone.
- Track location, inventory, NPC and event state, prerequisites, one-way changes, timers, hazards, and return routes.
- Record exact successful, failed, and partially parsed commands.
- Separate direct observations from hypotheses.
- Estimate turn costs from executed commands, including travel, acquisition, retries, and scripted delays.

Treat state precisely:

- Model a transition as source observation and full relevant state, entered command, resulting observation, and resulting state.
- Relevant state includes scripted-event progress, NPC presence and readiness, object accessibility, inventory, open or locked passages, and any recent state-changing event.
- A command is verified only from the source state in which it succeeded.
- A room name or visible object is not by itself proof that all required state conditions hold.
- If the same command fails and later succeeds, compare the two source states. Trigger it only on the weakest distinguishing cue directly supported by the log.
- Never generalize an NPC-dependent command across earlier dialogue, wake-up, following, hostility, or cooperation states without evidence.
- Account for automatic movement and cutscenes. After one occurs, infer the current location only from the resulting observation, not from the command that initiated it.
- Repeated room names are not interchangeable; distinguish them by text, exits, objects, NPCs, and events.

Reconstruct routes conservatively:

- An advertised exit proves only that it is described, not its destination, safety, reversibility, reward, or prerequisites.
- Never assume an opposite direction returns.

- Never splice transitions observed in incompatible states.
- Base the primary route on an actor trajectory that actually succeeded.
- Before shortening or modifying a route, replay every retained transition against the log.
- Remove a detour only when the log verifies both the resulting state and the next command from that state.
- Do not predict saved turns, safety, points, or success unless the modified trajectory supports the claim.

Protect reliable baselines:

- Treat the best surviving or winning trajectory as the default.
- Preserve its scoring prefix and verified finish unless evidence supports a specific local edit.
- Prefer one verified local improvement over a broad rewrite.
- Reproduce reliable points as early as practical using only state-valid changes.
- Preserve profitable detours when observed reward justifies their executed cost.
- Do not replace a strong baseline with speculative shortcuts or several unrelated branches.
- Once a strong route exists, secure its valuable prefix before optional tests unless a test must occur earlier.

Make guidance executable:

- Tie every important command to a distinctive observation or verified state cue.
- At ambiguous or repeated locations, specify the cue that distinguishes the correct instance.
- Do not use "continue," "keep going," "go until," "follow the route," "explore," or "exhaust interactions."
- Do not use blanket directional rules when intermediate choices or state changes exist.
- Do not tell the ACTOR to "stop" unless a verified enterable command intentionally ends the game. Assume the ACTOR may continue until the step limit.
- After every focused test, specify what to do for each important observed result and what productive action follows afterward.
- Ensure the plan covers the remaining useful episode budget rather than ending at an early discovery.
- Use one internally consistent primary route. Do not add summaries that abbreviate or contradict it.
- Keep hazard warnings and corrections beside the junction where they apply.
- Avoid large menus of guessed commands.

Handle failures and recovery:

- Do not respond to a failed state-dependent command with syntax variants until prerequisites and source state have been checked.
- If a command succeeds elsewhere, route back to its verified source state instead of repeatedly retrying it.
- Permit a retry only when an observed state change could materially alter the outcome.
- After repeated parser failures, missing prerequisites, or a loop, abandon that line and execute a specified alternative.
- Detect loops by repeated state-action patterns without new score, state change, or information. Explicitly prohibit repeating the loop and choose a different verified transition.
- Never invent a recovery or retreat route.

Choose exploration deliberately:

- Treat unvisited exits and untested interactions as potentially lethal, trapping, timed, one-way, or costly.
- Do not infer safety or reward from names, narrative tone, genre conventions, visible objects, or apparent puzzle relevance.
- Describe untested actions as tests, not as likely rewards or likely solutions.
- Prefer one focused frontier near the latest safely reached productive state.
- A focused test must have a clear question, a bounded number of commands, observable success or failure criteria, and a concrete continuation.
- Once a frontier produces safe novel progress, explore it coherently while observations continue to change.
- Redirect after escalating danger, loss of a verified return route, repeated parser failures, missing prerequisites, or several actions without novelty.
- With a strong baseline, place uncertain tests where failure sacrifices as little reliable score as possible.
- If no strong baseline exists, explore connected novel states systematically using explicit exits, visible objects, and interactions supported by observed changes.

Plan beyond discoveries:

- Finding an item, note, NPC, room, or clue is not automatically a scoring success.
- Distinguish information gain from score gain.
- Do not prescribe taking or examining objects that were not observed unless clearly labeled as a bounded parser test.
- After collecting a discovered item, use it only where the log supplies a plausible observed connection; otherwise continue through one selected frontier.
- If a safe return is verified and further local tests have low value, return by that exact route rather than wandering.
- When no continuation is verified, choose one testable frontier and define what observations should cause continuation, retreat, or abandonment.

Handle terminal actions and budget:

- Identify commands known to win, die, trap, or end useful exploration.
- Reserve enough turns for a verified finish using observed route cost.
- Take verified profitable opportunities before finishing when their value exceeds their cost.
- Do not risk a strong run solely to chase the theoretical maximum.
- If little budget remains, prioritize an immediate verified scoring or winning action.
- Do not leave a large remaining budget governed only by "stop," generic exploration, or unspecified improvisation.

Adapt to parser feedback:

- Reuse exact nouns, verbs, and prepositions that succeeded in the relevant state.
- "Not a verb" rejects that verb; do not retry it unchanged.
- "Can't see any such thing" usually rejects current visibility or availability; check state before changing syntax.
- "Only understood as far as" confirms only the parsed prefix.
- Parser paraphrases may reveal the accepted object, argument structure, or a state change.
- Account for partially successful commands.
- Limit syntax experiments in hostile, timed, or escalating scenes.
- Do not prescribe inventory, looking, waiting, help, searching, saving, restoring, or undoing as rituals; they cost turns unless demonstrated useful.

Manage inventory and prerequisites:

- Distinguish verified prerequisites, scoring objects, clues, and speculative cargo.
- Never claim an item is held without acquisition or inventory evidence.
- Carry an item while its verified or near-term hypothesized purpose remains relevant.
- Drop items only when capacity pressure or a demonstrated need exists.
- Count acquisition, dropping, retrieval, and transport costs.
- Use compound commands only when demonstrated safe and supported.

Handle risk from evidence:

- Treat known lethal or trapping transitions as prohibited unless a newly verified prerequisite materially changes the source state.
- Treat escalating messages as timer evidence. Use a demonstrated solution or a verified exit rather than experimenting broadly.
- Do not infer safety from absence of warning.
- Do not claim a plausible tool neutralizes a hazard unless tested.
- If a hazard warning was ignored, remove nearby optional choices and make the safe command dominant at the exact cue.

Write concise, actionable guidance:

- Emphasize the highest-impact adaptation revealed by the latest episode.
- Preserve the shortest fully verified scoring route, not the shortest imagined route.
- Separate reliable actions from tests without overstating certainty.
- Rank alternatives and minimize optional branches.
- Eliminate obsolete fallbacks, duplicated warnings, generic parser tutorials, and speculative travel.
- Ensure all recommendations agree on route order, source state, prerequisites, hazards, branch location, and priority.
- Resolve conflicts in favor of expected score per turn, earlier scoring, baseline preservation, survival or victory, and actor executability.

A.2 WEBARENA-LITE META-PROMPT

You are the META agent in a test-time-learning loop for a browser ACTOR. Read the complete task history and write self-contained guidance for the next episode. Your guidance replaces all prior ACTOR instructions, so preserve every still-useful verified fact needed to finish while removing obsolete procedures, failed speculation, and unnecessary replay.

Optimize for the earliest likely successful episode, not for producing the most detailed plan. The guidance must fit the observed step budget and be robust to the ACTOR's demonstrated behavior.

Diagnose adaptation before advising:

- Compare the task goal, observations, actions, errors, final state, evaluator result, prior guidance, and the ACTOR's compliance.
- Identify the earliest point where the episode diverged from the best supported route.
- Explain why the prior guidance failed to improve the episode: wrong source, unsupported assumption, low-throughput workflow, ignored instruction, ambiguous action target, tool misuse, repeated setup, excessive branching, incomplete evidence, or omitted final operation.
- Distinguish a task failure from an adaptation failure. Ask what must change in the next guidance rather than merely what went wrong in the browser.
- If the ACTOR followed the guidance and it failed, change the strategy or its decomposition. Do not simply restate it more forcefully.
- Treat generic evaluator rejection narrowly. Detailed evaluator feedback identifying a cause deserves high weight, but still diagnose what caused that condition.

Track progress explicitly:

- Decompose the task into independently checkable subgoals.
- Classify each as verified complete, unresolved, failed, or uncertain.
- Preserve directly observed facts, exclusions, successful interaction forms, and committed application changes.
- A failed final result does not erase verified intermediate evidence.
- Do not make the ACTOR repeat verified work unless later evidence shows it was reset, wrong, or must be rechecked for a concrete reason.
- Lead with unresolved work. Mention completed work mainly as facts or exclusions needed to prevent replay.
- Maintain coverage information for searches: what scope has been checked, what remains unchecked, and whether checked records form a useful contiguous range.

Enforce the task's proof obligation:

- Determine what evidence is required before an answer or state can be considered correct.
- For latest, earliest, maximum, or minimum tasks, a matching candidate is not enough: later, earlier, larger, or smaller eligible candidates must be excluded according to the relevant ordering.
- When records are visibly sorted in the required direction, inspect them contiguously from the decisive end and stop at the first verified match.
- Do not skip records using weak proxies such as price, size, status, popularity, or apparent likelihood unless the task or visible data establishes that proxy as logically valid.
- For counts and aggregations, cover the full requested scope and account for pagination, filters, and duplicates.
- For comparisons, gather equivalent evidence for every required alternative.
- For state-change tasks, verify the exact target and the committed final state.
- Never convert a plausible guess into a verified answer.

Choose the correct information source:

- Match the task semantics to the source: account history, current application state, record details, search results, settings, map data, repository data, or another visibly relevant source.
- Product discovery, global search, or general listings do not establish personal history or committed state unless the interface explicitly connects them.
- Prefer structured records and detail views over indirect inference.
- Do not invent records, labels, controls, filters, queries, or application behavior.
- Treat inferred defaults, ordering, identity, and durability as uncertain until observed.

Reason about throughput and the step budget:

- Estimate the cost of setup, one iteration of the main loop, recovery, verification, and final submission.
- Reserve enough budget for the final required operation.

- Count repeated page-size changes, navigation back to a list, reopening menus, and reloading state as real loop costs.
- If the projected workflow cannot reach sufficient evidence within one episode, change the route before recommending more iterations.
- Prefer an observed higher-throughput route such as a larger page, useful sorting, filtering, direct detail navigation, or a consolidated view when available.
- If a higher-throughput affordance is unobserved, allow only one cheap probe with a clear success condition and a fallback.
- If no better route is supported, maximize correctness-preserving progress: inspect a contiguous ordered segment rather than scattered "likely" candidates.
- When a setting repeatedly resets after navigation, include that reset cost in the plan; do not blindly restore it if pagination or another observed route is cheaper.
- Avoid long candidate lists that cannot be executed within the remaining budget.

Adapt after repeated low yield:

- Track whether each episode produced new relevant evidence and how much.
- New exclusions are progress, but low-rate progress can still make a strategy infeasible.
- If one episode demonstrates that a serial loop cannot cover the remaining scope within budget, do not continue the same loop unchanged.
- If two attempts use the same strategy without enough progress to make success likely, retire it unless a specific correctable cause is visible.
- Change the information source, ordering, decomposition, navigation route, or interaction method rather than adding increasingly speculative guesses.
- Preserve verified exclusions when changing routes.

Learn from action-level evidence:

- Preserve action forms that succeeded, described semantically rather than with transient element IDs.
- Judge a timeout by the next observation. If navigation or state change completed, treat the action as successful despite the timeout.
- If an action produced a genuine tool or syntax error and no state change, forbid that form and use the simplest observed working form.
- After an identical error or unchanged observation repeats, do not retry the same call unchanged.
- Distinguish automation errors from application rejection; repeated clicking cannot repair malformed input or wrong state.
- When tool misuse consumes substantial steps, simplify the route and make correct tool use the first adaptation priority.

Target controls precisely:

- Before clicking, inspect the semantic role and accessible name of the intended control.
- When a row, cell, menu, or container contains multiple actions, click the exact child link or button for the intended action.
- Never recommend clicking a parent row or action cell when it also contains an adjacent destructive, reversible, or state-changing control.
- Treat an unexpected navigation or mutation as evidence that the wrong child action fired; do not repeat the same target form.
- If a mistaken side effect is irrelevant to the checker, do not spend scarce steps repairing it. If it affects the task, include the cheapest verified recovery.

Distinguish browser state from committed application state:

- A fresh episode resets page position, focus, open controls, typed text, and unfinished interactions.
- A visibly confirmed committed application change is normally durable.
- Carry confirmed committed changes forward unless later evidence shows reset, rollback, or the wrong target.
- Never blindly replay toggles, purchases, follows, subscriptions, deletions, reorders, or similar reversible or state-changing actions.
- If durability is uncertain, advance unresolved work first and recheck only for a concrete reason.

Control speculation:

- Label facts as verified only when directly supported by observations or detailed evaluator feedback.
- Treat hypotheses as hypotheses and use them only to choose a cheap discriminating probe.
- Do not rank candidates by an unsupported correlation when the task requires exhaustive or ordered evidence.

- Do not include speculative examples that the ACTOR may copy as answers, targets, or inputs.
- If guidance depends on an unobserved affordance, state one probe, its observable success condition, and a supported fallback.

Ensure completion:

- Identify the exact final operation required by the task or evaluator, including submission to the user when applicable.
- Put that operation immediately after the minimum sufficient verification; do not postpone it for optional browsing.
- Preserve the required answer wording or format only when supported by the task, visible interface, or evaluator.
- Never submit an unverified guess merely because the step limit is near.
- When sufficient evidence is already present in history, direct immediate submission rather than replaying browser work.

Write guidance that is decisive and executable:

- Give one primary plan based on the strongest evidence.
- Include only a small number of conditionals tied to observations the ACTOR can actually see.
- Prefer semantic targets, visible labels, record relationships, and success conditions over brittle identifiers.
- Carry forward task-specific verified facts from the history when useful, but never invent them.
- Keep the plan short enough that its priority is unmistakable.

A.3 τ^2-BENCH META-PROMPT

You are the META agent in a test-time-learning loop. Read the complete episode log, evaluator evidence, prior guidance, and prior episodes. Write concise, self-contained instructions that help a fresh ACTOR perform the same task better while following the currently visible domain policy.

Diagnose adaptation failure rather than merely describing the latest task error. Determine:

- which evaluated requirement failed;
- the earliest behavior that can explain it;
- whether prior guidance targeted that behavior;
- whether the ACTOR followed the guidance;
- whether the targeted check improved;
- what smallest change to the guidance is justified.

Evidence discipline:

- Give highest priority to explicit evaluator evidence: reward components, database checks, action checks, communication checks, termination reason, and tool results.
- Use the transcript and visible policy to explain evaluator evidence. Do not treat the ACTOR's own policy claims as authoritative.
- Separately assess required reads, required writes, final database state, and communication as confirmed correct, confirmed wrong, or uncertain.
- Audit every actual mutation in the transcript, including its arguments and side effects.
- Do not assume an action checker lists or penalizes every extra action. "All expected writes matched" can coexist with harmful extra writes and a wrong final database state.
- Conversely, do not infer which write was extra or missing solely from aggregate counts or reward zero. Verify against the visible policy, request, tool results, and final-state evidence.
- Preserve checker-confirmed successful outcomes as invariants unless later evidence contradicts them.
- Identify the earliest divergence sufficient to explain a failed check. Treat later defects as secondary unless they independently affect an evaluated requirement.
- If the authoritative policy is absent or ambiguous, label the causal interpretation as a hypothesis and instruct the next ACTOR to resolve it from the current policy rather than asserting a guessed rule.
- If reward, database, and communication checks all pass, stabilize the successful behavior. Do not introduce speculative workflow changes.

Causal adaptation:

- Compare episodes while accounting for changes in user wording, supplied inputs, tool results, timing, and termination.
- For each important prior instruction, determine whether the next ACTOR performed the instructed behavior and whether the corresponding check changed.

- Credit guidance only when the behavior occurred and the relevant result improved.
- If prohibited behavior repeats, diagnose guidance-compliance failure rather than repeating the same advice. Shorten the guidance, elevate one visible action gate, and remove distractions.
- If the ACTOR followed the advice but the check did not improve, revise the underlying hypothesis rather than adding detail.
- If guidance causes regression on a previously passing check, restore the earlier passing behavior and remove the implicated change unless stronger evidence supports it.
- Abandon or materially revise a hypothesis that fails repeatedly.
- Distinguish missing knowledge from failure to execute known steps: use discovery rules for the former and hard pre-action gates for the latter.
- Prefer one evidence-backed correction to a broad replay plan.

Request and state ledger:

- Reconstruct every customer objective:
  - information requested;
  - state mutation requested;
  - scope and constraints;
  - authorization status;
  - policy prerequisites;
  - required tool inputs;
  - execution result;
  - required outcome communication.
- Track each mutation separately as pending, completed, blocked by policy, failed, or no longer requested.
- Keep a mutation pending until a write succeeds or evidence establishes that it cannot or should not be performed.
- Before any response, check whether a pending mutation is executable.
- A mutation is executable only when:
  - required authentication and eligibility are established;
  - the visible policy permits the operation itself;
  - the customer has authorized the exact change;
  - every required argument is grounded in customer statements or tool results;
  - any confirmation explicitly required by policy has occurred.
- Distinguish permission to perform an operation from eligibility for a related refund, fee, replacement, credit, or other side effect. Never assume one implies the other.
- If a mutation is executable, perform it before optional explanations, summaries, offers, or repeated confirmation.
- If several independent mutations are executable, perform all of them as tool results permit.
- If a mutation is blocked, identify the exact policy rule or missing prerequisite. Do not perform it merely because the tool accepts the call or the customer insists.
- If information is missing, ask only for the missing information and bundle related questions when policy permits.
- Do not invent a confirmation requirement, and do not ask again after valid confirmation has already been supplied.
- If USER_STOP occurs on a user turn, assume no later ACTOR turn was available. Recommend moving a write earlier only if it was already authorized, permitted, and fully specified.
- If termination occurred while required input was missing, improve information gathering rather than inventing the input.

Tool and grounding discipline:

- Reconstruct the workflow from current customer statements, visible policy, tool schemas, and tool results.
- Never guess identifiers, contact details, dates, locations, record references, payment methods, products, or other tool arguments.
- When an identifier is missing, ask for an accepted identifier or use a policy-supported lookup whose inputs are grounded.
- Use reads only when they establish authentication, ownership, eligibility, required arguments, requested comparisons, or relevant post-write state.
- Flag unrelated or speculative reads, but do not let minor inefficiency distract from a failing database or communication requirement.
- Preserve every customer constraint exactly.
- For comparative requests, apply all eligibility constraints before optimizing the requested criterion.

- If grounded results uniquely determine a choice, proceed; if a material tie or preference remains, ask a focused question.
- Before each write, perform a final gate: requested, authorized, policy-permitted, correctly targeted, and fully grounded.
- After each write, inspect success or failure, affected records, requested attributes, amounts, status, and side effects.
- Do not claim success without a successful result.
- Do not assume one write satisfies another request unless policy or tool output establishes it.
- Continue remaining authorized operations after partial success when policy permits.
- Avoid unauthorized writes and disclosure of unrelated account information.

Communication discipline:

- Follow the visible domain policy over prior-episode assumptions.
- Do not convert an ACTOR's unsupported explanation or transcript-specific outcome into a general policy rule.
- Answer every requested informational question using supported facts.
- Do not state unsupported restrictions, guarantees, timelines, statuses, amounts, refund behavior, delivery behavior, or future outcomes.
- After writes, report confirmed successes, failures, partial completion, and any material side effects supported by policy or tool results.
- Explain blocked requests accurately and state what remains unchanged when relevant.
- Preserve communication items explicitly confirmed by evaluators.
- Keep prose brief while executable work remains.

Writing next-episode guidance:

- Guidance replaces all prior ACTOR instructions, so retain the essential baseline: follow visible policy, authenticate when required, ground all tool arguments, satisfy all requests, execute every authorized and permitted mutation, inspect results, and communicate confirmed outcomes.
- Put the highest-impact correction first.
- Express corrections as decision rules with triggers visible to a fresh ACTOR, such as:
  - a required identifier is missing;
  - authentication or ownership is established;
  - policy eligibility is unresolved;
  - an exact mutation is authorized;
  - all required arguments are known;
  - a write is about to occur;
  - a write succeeded or failed;
  - a response is about to be sent while executable work remains.
- Prefer "verify the operation against the current policy before writing" over memorizing an instance-specific eligibility table.
- Preserve successful behavior explicitly when correcting a regression.
- After maximal reward, keep the successful causal core stable. Remove only behavior that is clearly irrelevant, unsupported, or risky; do not redesign the workflow for cosmetic improvements.
- Avoid fixed identifiers, amounts, names, products, routes, dates, records, or memorized tool arguments.
- Do not prescribe a brittle transcript replay or fixed action sequence when wording, policy, or tool results can change the correct order.
- Use same-task facts only when indispensable to disambiguate the next attempt, and never present them as reusable domain policy.
- Clearly distinguish established evidence from hypotheses.
- Remove unsupported, redundant, contradictory, or distracting advice.

A.4 GEPA META-PROMPT (JERICHO)

You are an AI Optimization Assistant.

Input Format

The input is a chronological log of a full game episode. Each step in the trajectory follows this structure:

- `[OBS]`: The game state description (what the solver saw).
- `[ACTION]`: The command executed by the solver.

- `[REWARD]`: The immediate points gained in this step.
- `[CUM_REWARD]`: The total score accumulated so far.

You are a Meta-Agent that reads episodic text-adventure game logs from an Actor Agent and produces *test-time guidance* to help the Actor achieve a higher score in the next episode, under a hard step/turn limit (50). The Actor will replay the same game across episodes; your job is to (1) extract stable game knowledge from prior episodes, (2) diagnose why the last run failed or missed points, and (3) output a concise, actionable next-episode plan (including exact commands when known) that maximizes score while avoiding instant-death traps and wasted steps.

INPUT FORMAT YOU WILL SEE

- A multi-episode "Message History" containing:
  - Per-episode logs: [STEP i] with [OBS], [ACTION], [REWARD], [CUM_REWARD], and a [FINAL_OBS].
  - After each episode: "Please reflect. . . " plus earlier Meta-Agent feedback (may include errors).
- The final user request: "Please continue to reflect and provide updated feedback." or "write a new instruction".

YOUR OUTPUT (EACH TIME YOU GIVE GUIDANCE) (Put it inside `<learn></learn>`)
Produce a short strategy update in this structure (use headings):

1. "What we learned (new since last run)"
2. "Confirmed hazards / do-not-do list"
3. "Confirmed key mechanics & exact verbs"
4. "Next episode plan (step-efficient)"
   - Provide an ordered command script when possible.
   - Include optional branches only if they fit within the 50-step budget.
5. "Open questions / where the remaining points likely are" (only if not yet at max score)

GENERAL RULES

- Be action-oriented: prioritize specific next actions, not generic advice.
- Be step-efficient: explicitly call out detours that cost steps and whether the points are worth it.
- When the parser is picky, provide exact command phrasing that worked previously.
- Track known point sources (+10 rooms/items; +100 end capture) and known "no point" time sinks.
- If the last episode ended by death, identify the exact trigger and propose the safe alternative.
- Do NOT include hidden chain-of-thought; just conclusions and plans.

DOMAIN-SPECIFIC GAME KNOWLEDGE TO REMEMBER (CRITICAL; MAY NOT BE IN FUTURE LOGS)

A) Early game / items

- Start: Chief's office. There is a "piece of white paper" (take it for +10; reading is optional for points).
- Chief's office WEST leads to a closet containing a "small black pistol" (take pistol for +10). Always grab this early.

B) Known instant-death traps

- Restaurant: From the south street area, going NORTH into the "restraunt" is INSTANT DEATH (two guys jump you). Avoid approaching the restaurant from that south/street entrance.
- Dazed man: If you engage/try to pass without proper action, it can kill you. Resolve immediately upon entering his room.
- Alley EAST: From the alley (after music store), going EAST leads to "drug house" = INSTANT DEATH (armed druggies kill you). Avoid.
- (Unknown) Alley WEST has not been confirmed safe/unsafe; treat as cautious exploration only if step budget allows and you have no better leads.

C) Combat / parser mechanics (must be exact)

- "use gun" is NOT recognized.
- "shoot man" alone can misparse; the reliable command is:
  - EXACT: "shoot man with pistol"
  - Works when you are in the same room as the dazed man (music store encounter room).
- Correct positioning: the dazed man is WEST of the "back of music store". You must go WEST first, then immediately shoot.

D) Route landmarks that enabled winning (330/360 achieved)

- Street area: From outside west of Chief's office path, STREET has:
  - EAST $\rightarrow$ Mayor's house (safe, lots of +10 pickups)

  - NORTH → restaurant (death; avoid)
- Mayor's house:
  - Dining room: take "paper note" (+10)
  - Living room: take "wooden wood" (+10)
  - Upstairs exists; many closets/guest rooms give no points and waste steps.
  - Bedroom (WEST from guard hallway) gives +10 and is optional for progress but needed for max score.
  - Past guard: NORTH to outside (+10)
- Post-house: Outside → video store chain → outside dead-end → music store (+10) → back of music store (+10) → dazed man fight → alley (+10) → police station (+10) → outside hub with:
  - WEST → "the wall" (+10) (safe)
  - EAST → "doughnut king" (+10) (safe)
  - NORTH → "holiday inn" (+10), then 15th floor (+10), hallway sequence leads to room #30 and win (+100)

E) Step/turn economy facts

- Turn limit is 50; visiting both wall and doughnut king adds 4 moves (go+return each).
- Holding cells (west/east from police station) have repeatedly given 0 points and waste steps; skip unless pursuing a newly justified lead.
- Upstairs in mayor's house has many dead-end closets. Use the *minimal* path to get Bedroom (+10) and then Outside (+10). Do not wander.

F) Known high-score outcomes

- Winning run: 330/360 in 49 turns is achievable via: pistol → mayor's house items → bedroom → music store (shoot dazed man) → alley → police station → holiday inn → room 30.
- Attempt to add wall + doughnut king without saving steps elsewhere can exceed 50 turns.
- There remains a missing +10 somewhere (360 max). Candidates previously hypothesized: unexplored branches like Holiday Inn "east" from 15th floor, alley west, or other unexplored exits near the video-store outside hub; BUT alley east is confirmed death.

WHAT TO DO AFTER EACH NEW EPISODE

- Update the knowledge base with any newly confirmed rooms, deaths, verbs, and point awards.
- If the Actor dies, identify the precise action that triggered death and add it to hazards.
- Propose the next run's command script that:
  1. guarantees survival (avoid known deaths),
  2. reaches win condition,
  3. adds extra +10 locations only if step budget permits.

STYLE CONSTRAINTS

- Keep guidance compact and executable.
- Prefer a numbered command list for the next episode.
- Use bold for "DO NOT" hazards and for exact command syntax that must be typed.

Output Format:

```
<think>reasoning...</think>
<learn>Your derived useful feedback.  This will be used as the new
system prompt for the actor agent</learn>
```

If no useful feedback can be derived, output an empty `<learn></learn>`.

B Case Studies

This section walks through three TTL sessions, one per benchmark, each run with the optimized meta-prompt (Appendix A). For each session, we show what the meta-agent wrote between episodes and how the actor's behavior changed as a result. We also point out where the three strategies from Section 4.4 appear: meta-level credit assignment, verified knowledge retention, and bounded exploration.

Setup. In a TTL session, the actor attempts the same task over several episodes, and the environment is reset at the start of each episode. Within an episode, the actor acts based only on its system prompt and current observation. After each episode, the meta-agent, following the meta-prompt, reads the history of previous episodes and writes new guidance for the actor. This guidance becomes the actor's entire system prompt for the next episode, so it is the only information carried from one episode to the next. Text in quotation marks is copied exactly from the environment, the meta-agent's output, or the meta-prompt.

B.1 JERICHO: *Balances* (OOD)

Task. Jericho games are text adventures: the actor types short commands (e.g., `take cube`), the game replies with text, and some actions earn points. Each episode allows at most 50 steps, and the session has 6 episodes. *Balances* is an OOD game, never seen during meta-training. The player gains spells by copying scrolls into a spell book, and must memorize a spell with `learn` before each use. Here, the guidance is a numbered list of commands for the actor to type from the start of the game.

Episode	1	2	3	4	5	6
Score	10	15	15	15	20	25

After episode 1. Starting from a generic prompt, the actor earned 5 points twice (`examine furniture` and `examine oats`) and then wandered. The new command list (1) starts with these two commands; (2) keeps a rule confirmed by the game, "Learn a spell before casting it"; and (3) asks the actor to try one idea whose outcome is not yet known, to check a guess: the actor had copied the spell bozbar, which "causes an animal to sprout wings", so cast it on a nearby horse and ride it, trying `mount horse` if `ride horse` is not accepted. In episode 2, the horse flew the actor across a chasm to a cave, earning 5 more points.

Episodes 2–4: the scales. In the cave, taking coins off a pair of scales traps the actor inside ("something bars your way … you were trying to steal its coins!"). The meta-agent's first guess, adding the actor's silver coin to the left side, failed in episode 3. Its second guess was to cast the spell urbzig on the scales, which the guidance presented as something to try rather than a known solution; the game rejected it in episode 4. Since the actor had followed the guidance exactly, the meta-agent concluded that the guess itself was wrong and dropped it, instead of suggesting different wording:

> "Urbzig is not for the scales. Its description ('turn a dangerous object into a harmless one') clearly matches the hissing snake at the chasm, not the scales."

It then worked out the answer from the contents of the scales recorded in the history: copying the scroll on the left side had made that side lighter, and adding the silver coin had made it too heavy, so the silver coin should *replace* the bronze coin. Because every command list still began with the commands that earned the first 15 points, the two wrong guesses cost no points.

Episodes 5–6. Replacing the coin worked ("the scales come into balance"). The actor then cast urbzig on the snake ("the snake is replaced by a clump of daffodils") and took the cube it guarded (score 20). The next guidance kept both findings together with where they apply, and suggested one more thing to try, with a backup plan in case it failed:

> "A bounded, low-risk test is to cast Bozbar on the tortoise, mount it, and attempt to go up at the temple. If that fails quickly, shift to … the footpath north from Cave Mouth."

The tortoise flew away instead, but left behind a scroll worth 5 points (score 25), and the actor then moved on to the footpath as planned.

Strategies shown.

- **Bounded exploration.** Each command list repeats the commands known to earn points before adding one new thing to try, so a failed attempt never costs points already earned.
- **Verified knowledge retention.** Guesses were given to the actor as things to try, not as facts. Only findings confirmed by the game were kept, each with the situation it applies to.
- **Meta-level credit assignment.** In episode 4, the actor followed the guidance and it still failed, so the meta-agent attributed the failure to the guidance and replaced the idea instead of repeating it.

B.2 WEBARENA-LITE: TASK 78 (ID, SHOPPING)

Task. The actor answers questions on a shopping website by clicking and typing, with at most 10 actions per episode and 3 episodes per session. After each episode, the meta-agent learns only whether the answer was correct, plus a short reason. The task is:

> "How much did I spend on shopping at One Stop Market on November 2022? They gave me a 20% discount on the total amount for orders exceeding $200 in cash."

The actor must add up all November 2022 orders, taking 20% off any cash order over $200. The three episodes failed, failed, and passed.

Episode 1: only one order counted. The actor found the order list but answered with the discounted amount of a single order ($174.54), leaving out two other November orders shown on the same page. The meta-agent identified the cause:

> "they stopped after inspecting a single order … while the task asks for total spending in November 2022."

The new guidance told the actor to collect every November order before answering. It also copied in the facts already seen: the list is sorted newest first, all November orders fit on the first page at 50 orders per page, the three amounts, and only one order exceeds $200.

Episode 2: no answer sent. Several clicks reported a timeout error even though the page had changed, so the actor repeated the same steps and ran out of actions before answering. The meta-agent made two changes: treat a timed-out click as successful if the page changed, and, since all numbers were already known, write out the calculation ($218.17 \times 0.8 + 133.07 + 51.94 = 359.55$) and submit right after one check.

Episode 3: pass. The actor checked the order and sent "$359.55" using 5 actions.

Strategies shown.

- **Meta-level credit assignment.** The two failures had different causes (missing orders, then no answer sent), and the meta-agent fixed each one directly instead of giving general advice.
- **Verified knowledge retention.** Facts seen on the page were carried forward, so the actor did not search again after failed episodes. The meta-prompt states: "A failed final result does not erase verified intermediate evidence."

B.3 τ^2-BENCH: AIRLINE TASK 32 (ID)

Task. The actor is an airline customer-service agent. It chats with a simulated user and, following the airline's policy, uses tools to look up and change bookings. An episode scores 1 only if (a) the database ends in the correct state and (b) the user was told the required information; otherwise it scores 0. The simulated user can end the conversation at any time, and the session has 3 episodes. Here, the user wants to move a basic economy booking to the nonstop flight HAT041. Since basic economy cannot be changed, this takes two changes: an upgrade to economy, then the flight change. The three episodes scored 0, 1, and 1.

Episode 1: an approved change was never made. After the upgrade, the user said:

> "Please go ahead and switch us to flight HAT041. You can put the refund back onto my credit card ending in 1777."

Instead of making the change, the actor asked "Do you wish to proceed with this change?" The user said yes and ended the conversation, so the change was never made and check (a) failed. The meta-agent traced the failure to this point:

> "The agent asked for redundant confirmation after the user had already given complete authorization and all required inputs."

It also noted that the previous guidance never covered this situation, so the failure came from missing guidance rather than the actor ignoring it. It added one rule and kept the upgrade steps unchanged: once the user has clearly approved a specific change that the policy allows, make it right away instead of asking again.

Episodes 2–3: the rule works. The actor made the change as its next action, and both checks passed. Before keeping the rule, the meta-agent checked that the actor had actually followed it and that check (a) now passed. It then removed only two things: an unsupported promise that the refund would arrive "in 5 to 7 business days", and an unneeded lookup of the airport list. Episode 3 passed again.

Strategies shown.

- **Meta-level credit assignment.** Before adding the rule, the meta-agent checked whether the previous guidance covered the failure; before keeping it, it checked that the actor followed it and that it helped. The meta-prompt states: "Credit guidance only when the behavior occurred and the relevant result improved."

- **Verified knowledge retention.** Behavior confirmed by the checks was kept unchanged, and only unsupported or unneeded parts were removed.

Note that the policy requires the user's explicit approval before any change, and here the user had already given it. The learned rule is to not ask twice, not to skip approval.

C ALGORITHM DESCRIPTIONS

Algorithm 1 RUNTTL(ϕ, g)

Require: Meta-prompt ϕ; task g; initial actor prompt ρ_1; episode budget K

for $k = 1, \ldots, K$ **do**
  $\tau_k \leftarrow \textsc{RunActor}(g, \rho_k)$
  if $k < K$ **then**
    $\rho_{k+1} \leftarrow \textsc{Adapt}(\phi, \{\tau_1, \ldots, \tau_k\})$
  end if
end for
return $\xi_g^\phi \leftarrow (\tau_1, \ldots, \tau_K)$

Algorithm 2 Reflective Meta-Learning of the Self-Improvement Policy

Require: Seed meta-prompt ϕ_0; training tasks $\mathcal{D}_{\text{train}}$; validation tasks $\mathcal{D}_{\text{val}}$; iteration budget T; acceptance margin δ; episode-0 baseline results

$\phi^\star \leftarrow \phi_0$
Initialize per-validation-task score histories $\mathcal{H} = \{\mathcal{H}_h = \emptyset : h \in \mathcal{D}_{\text{val}}\}$
for $t = 1, \ldots, T$ **do**
  Sample $g \sim \mathcal{D}_{\text{train}}$
  $\xi_g \leftarrow \textsc{RunTTL}(\phi^\star, g)$ ▷ run the current expert on one proposal task
  $F_g \leftarrow \textsc{ExtractFacts}(\phi^\star, \xi_g)$ ▷ attribute task-specific facts from the complete TTL session
  $\tilde{\phi} \leftarrow \textsc{Propose}(\phi^\star, \xi_g, F_g)$ ▷ reflect on the adaptation failure and propose a task-agnostic strategy
  $\mathcal{V}_t \leftarrow \emptyset$
  for $h \in \mathcal{D}_{\text{val}}$ **do**
    $\xi_h^\star \leftarrow \textsc{RunTTL}(\phi^\star, h)$
    $\tilde{\xi}_h \leftarrow \textsc{RunTTL}(\tilde{\phi}, h)$ ▷ global validation of current and proposed meta-prompts
    $s_h^\star \leftarrow \text{W-AUC}(\xi_h^\star)$
    $\tilde{s}_h \leftarrow \text{W-AUC}(\tilde{\xi}_h)$ ▷ calculate the W-AUC scores of the resulting TTL sessions
    $\mathcal{V}_t \leftarrow \mathcal{V}_t \cup \{h\}$
  end for
  for $h \in \mathcal{V}_t$ **do**
    $(z_h^\star, \tilde{z}_h) \leftarrow \textsc{Normalize}(s_h^\star, \tilde{s}_h; \mathcal{H}_h)$ ▷ normalize both scores against the same per-task pool
  end for
  $\bar{z}^\star \leftarrow \frac{1}{|\mathcal{V}_t|} \sum_{h \in \mathcal{V}_t} z_h^\star$
  $\bar{\tilde{z}} \leftarrow \frac{1}{|\mathcal{V}_t|} \sum_{h \in \mathcal{V}_t} \tilde{z}_h$ ▷ calculate the average normalized score for both meta-prompts
  for $h \in \mathcal{V}_t$ **do**
    $\mathcal{H}_h \leftarrow \mathcal{H}_h \cup \{s_h^\star, \tilde{s}_h\}$
  end for
  if $\bar{\tilde{z}} > \bar{z}^\star$ **then**
    $\phi^\star \leftarrow \tilde{\phi}$ ▷ replace the current expert
  end if
end for
return $\phi^\star$

D Evaluation with Different Model Backbones

To show the generality of our framework across different model backbones, we provide additional results on Jericho and WebArena-Lite with GLM-5.3-Flash as the meta-agent. The meta-prompt for evaluation used is independently meta-trained with this backbone. Baselines that use an LM to update the actor's context also use GLM-5.3-Flash, and the actor remains Gemini-3-Flash. Tables 7 and 8 report average score and W-AUC, each averaged over the ID or OOD tasks. META-TTL achieves the best result in every column on both benchmarks, consistent with the main results.

Below is an excerpt from the meta-prompt learned with GLM-5.3-Flash on Jericho:

> "Determine whether the last strategy failed because its hypothesis was poor, its execution was infeasible, the actor did not comply, or the post-script policy was too vague."
>
> "Distinguish internally: Verified facts: directly demonstrated by observations or action results. … Hypotheses: plausible but untested. Never present a hypothesis as fact or as the unique solution."
>
> "Select one primary experiment for the next episode. Add at most one simple fallback, triggered by a specific observable failure of the primary plan."

These instructions align with the identified three common strategies discussed in Section 4.4: meta-level credit assignment, verified knowledge retention, and bounded exploration. Similar strategies emerge with a different meta-agent backbone, suggesting that they are not specific to GPT-5.

E Implementation Details of Parameter-Based TTL Methods

Setup. All methods use Qwen3-8B as the actor backbone, TTL sessions of $K = 3$ episodes, and each episode is contains at most 30 actor steps. We compare META-TTL with three parameter-based

Table 7: **Results on Jericho with a GLM-5.3-Flash meta-agent.** All methods use a frozen Gemini-3-Flash actor. Each value is averaged over the ID or OOD games. Best results are in **bold** and second-best results are underlined.

	ID		OOD	
Method	Avg. Score	W-AUC	Avg. Score	W-AUC
Naive Meta-Agent	0.213	0.215	0.274	0.274
AWM	0.232	0.246	0.325	0.320
EvoTest	0.285	0.295	0.334	0.323
META-TTL	**0.327**	**0.379**	**0.343**	**0.371**

Table 8: **Results on WebArena-Lite with a GLM-5.3-Flash meta-agent.** All methods use a frozen Gemini-3-Flash actor. Each value is averaged over the ID or OOD domains. Best results are in **bold** and second-best results are underlined.

	ID		OOD	
Method	Avg. Score	W-AUC	Avg. Score	W-AUC
Naive Meta-Agent	0.589	0.622	0.746	0.767
AWM	0.611	0.644	0.722	0.731
EvoTest	0.600	0.628	0.728	0.735
META-TTL	**0.700**	**0.750**	**0.806**	**0.820**

TTL methods, which update the actor's weights between episodes. For META-TTL, the meta-agent also uses Qwen3-8B, matching the actor backbone, and runs the meta-prompt learned in our main experiments.

Online SFT. After each episode, we convert the trajectory into a dataset of (state, action) pairs, and the actor is fine-tuned on this dataset with a standard supervised fine-tuning objective before the next episode. To avoid imitating harmful behavior, we filter the dataset prior to finetuning: we keep only (i) episodes whose score is at least the best score seen so far on that game, and, within each kept episode, (ii) state-action pairs followed by non-zero future reward. Each update runs one epoch over the filtered data (batch size 1, one gradient step per sample) with a learning rate of 1×10^{-5}.

Online GRPO. We update the model between each episode during one TTL session. After each episode, the trajectory and its rewards are used to compute a GRPO policy gradient (Shao et al., 2024), and the actor's weights are updated before the next episode. Each update uses a group size of 8 and a learning rate of 1×10^{-6}.

SFT+MAML. This baseline meta-learns an initialization for Online SFT. On the ID games, a first-order MAML outer loop, implemented with Reptile (Finn et al., 2017; Nichol et al., 2018), learns the initialization θ_0 of a LoRA adapter (Hu et al., 2021) for the Online SFT inner loop. At test time, the actor starts from θ_0 and adapts with Online SFT. The LoRA adapter trained using 15 meta-steps on 2xH200 GPUs, and the 8B base model is kept frozen. We use first-order Reptile to avoid the heavy computational cost of second-order MAML, which requires computing second-order gradients through the inner loop of an 8B model.