LEARNING TO LEARN-AT-TEST-TIME: Language Agents with Learnable Adaptation Policies

Zhanzhi Lou [1] **Hui Chen** [1] **Yibo Li** [1] **Qian Wang** [1] **Bryan Hooi** [1]

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 an adaptation policy that updates the actor policy based on experience from previous episodes, thereby improving future behavior. Existing methods rely on fixed, hand-crafted adaptation policies rather than optimizing them for downstream improvement. We argue that optimal adaptation 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 adaptation policies as a bi-level optimization problem. Within this framework, the inner loop executes the standard TTL process, measuring how effectively a candidate adaptation policy helps an agent correct errors across sequential episodes. Guided by the agent's performance, the outer loop employs evolutionary search over a diverse distribution of training tasks to continually optimize the adaptation policy. We evaluate META-TTL on Jericho and WebArena-Lite across both in-distribution (ID) and out-of-distribution (OOD) settings, using multiple meta-agent backbones. Results on both benchmarks show that META-TTL consistently outperforms hand-crafted baselines, suggesting that the optimized adaptation policy encodes transferable strategies that generalize beyond the training task distribution.

[1]National University of Singapore. Correspondence to: Bryan Hooi, Hui Chen and Zhanzhi Lou <{hui.chen,dcsbhk}@nus.edu.sg, zzlou@u.nus.edu>.

Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026. Copyright 2026 by the author(s).

1. Introduction

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 an *adaptation policy* that updates the actor policy based on accumulated experience. Unlike the actor policy, which determines the agent's behavior within an episode, the adaptation policy determines how the actor policy evolves across episodes. However, most existing methods, such as Reflexion (Shinn et al., 2023), perform adaptation by relying purely on the pretrained capabilities of the underlying LLM. Fundamentally, the adaptation 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 adaptation 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 adaptation policy from task environments by optimizing it for downstream improvement at test time.

To this end, we propose **META-TTL**, a framework that casts TTL as a meta-learning problem: given a distribution of training tasks, we formulate the discovery of effective adaptation policies as a bi-level optimization. Concretely,

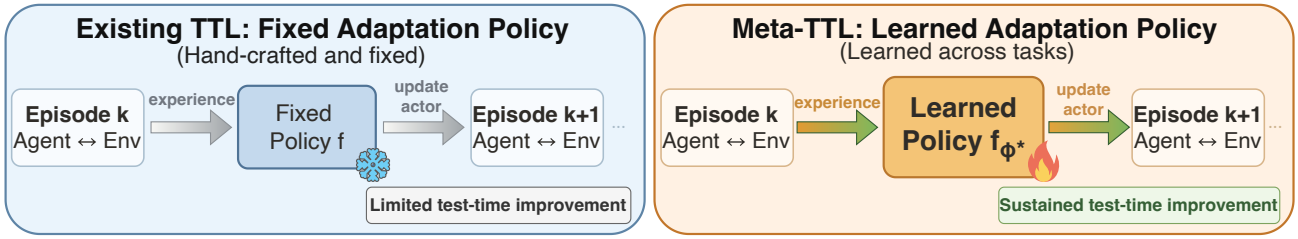


Figure 1. Adaptation policies determine how the agent uses its experience up to episode k to update the actor before episode $k+1$. Existing methods use a fixed adaptation rule, whereas META-TTL learns the adaptation policy across tasks and applies it zero-shot at test time.

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 adaptation policy ϕ helps the agent improve across episodes. In the outer loop, we optimize ϕ over a distribution of training tasks through evolutionary search: we iteratively evolve candidate policies, evaluate them through the inner loop, and retain those that produce stronger TTL performance. At test time, the learned adaptation policy is frozen and applied zero-shot to unseen tasks.

A key distinction from prior work lies in what is being optimized (Figure 1). Existing TTL methods treat the adaptation mechanism—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 adaptation mechanism itself as the object of optimization: META-TTL learns across a distribution of training tasks how to adapt effectively, and deploys the resulting adaptation policy zero-shot at test time.

Our contributions are as follows:

- We formalize Test-Time Learning as a **meta-learning problem over adaptation policies**, providing a principled framework for optimizing how agents update themselves across episodes for self-improvement.
- We propose **META-TTL**, which uses evolutionary optimization on a task distribution to learn an adaptation policy that generalizes to unseen environments. In our instantiation, this policy is realized as a natural-language meta-prompt that turns generic self-correction into concrete adaptation instructions.
- We evaluate our framework on two language-based sequential decision-making settings and demonstrate that META-TTL significantly outperforms heuristic TTL baselines on both in-distribution and out-of-distribution tasks, achieving $\sim$ 120% improvement in average game score on Jericho ID ($50.4 \rightarrow 110.8$) and up to $\sim 15\%$ relative improvement in task success rate on WebArena-Lite ID ($0.55 \rightarrow 0.63$), with both gains generalizing to out-of-distribution tasks, validating that the learned adaptation policy acquires transferable adaptation strategies that generalize to unseen environments.

2. Related Work

Test-Time Learning Test-Time Learning (TTL) improves post-deployment performance through additional computation during deployment (Jiang et al., 2026a). Existing methods fall into two broad groups. **Gradient-based** methods update model weights at test time, either by fine-tuning on training examples (Akyürek et al., 2025; Acikgoz et al., 2025; Zweiger et al., 2025; Ye et al., 2026) or by test-time reinforcement learning (Zuo et al., 2025; Yuksekgonul et al., 2026). **Weight-frozen** methods keep model parameters fixed and perform adaptation in external state. Early work adopts the verbal reinforcement learning paradigm (Shinn et al., 2023; Madaan et al., 2023), while later methods store experience in memory to enable persistent adaptation (Wang et al., 2024; Wei et al., 2025; Chhikara et al., 2025; Suzgun et al., 2025; Zhou et al., 2025; Xu et al., 2025). Other work improves by learning the rules of a new environment through interaction (Chen et al., 2026a; Zhang et al., 2025), and EvoTest expands the scope further by evolving the agent configuration as a whole (He et al., 2025).

LLMs as Optimizers & Prompt Evolution A growing body of work studies how LLMs can search over natural-language instructions. Early approaches such as APE (Zhou et al., 2023) and OPRO (Yang et al., 2024) generate and score candidate prompts iteratively, while evolutionary variants such as PromptBreeder (Fernando et al., 2023), EvoPrompt (Guo et al., 2025) and ReEvo (Ye et al., 2024) introduce population-based search with mutation, crossover, and reflective feedback. More recent work scales these ideas: GEPA (Agrawal et al., 2026) applies Pareto-based selection to compound AI systems, AlphaEvolve (Novikov et al., 2025) targets scientific discovery, and EvoX (Liu et al.,

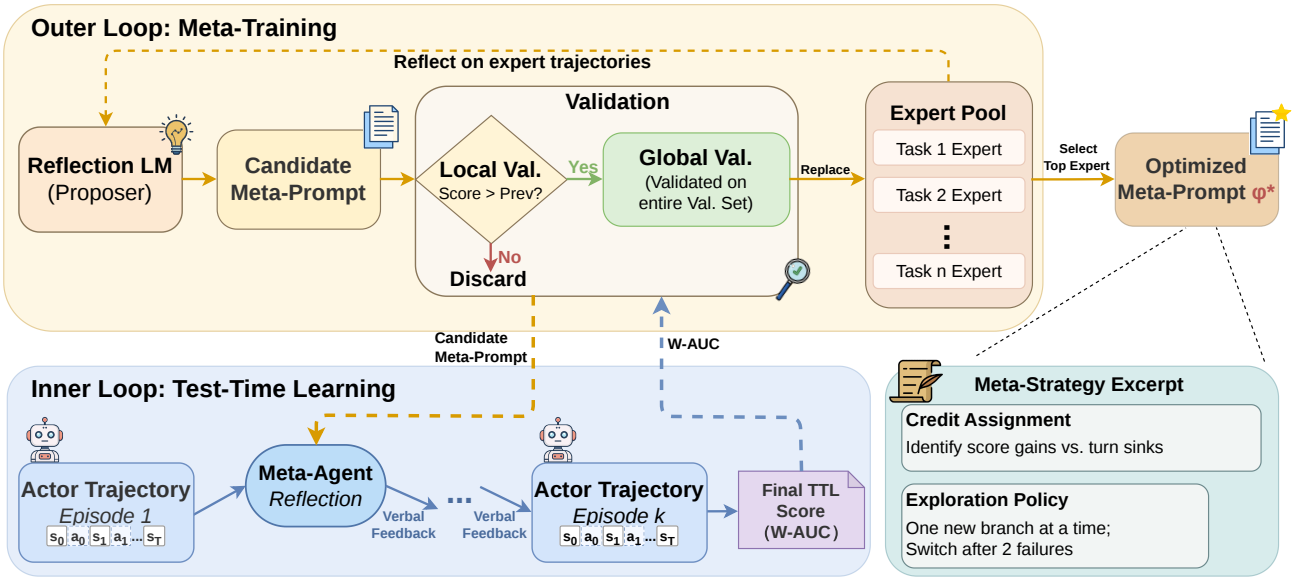


Figure 2. Overview of META-TTL. **Outer loop (meta-training):** A proposer LM reflects and proposes candidate meta-prompts, which are validated locally and globally before entering a per-task expert pool. After training, a single optimized meta-prompt ϕ^* is selected from this pool. **Inner loop (test-time learning):** The meta-agent, governed by ϕ^*, observes the actor's trajectory after each episode and generates verbal feedback that rewrites the actor's system prompt for the next attempt.

2026) co-evolves solutions and search strategies. On the systems side, TextGrad (Yuksekgonul et al., 2024), MetaReflection (Gupta et al., 2024) and ACE (Zhang et al., 2026) optimize compound pipelines through natural-language critiques or aggregated trial reflections. These methods show that experience on a training distribution can be distilled into improved prompts through offline optimization.

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 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 adaptation policies for sequential environments. Several concurrent works explicitly 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 a portable text artifact that transfers across backbones without retraining.

3. Methodology

We present **META-TTL**, a bi-level framework for learning a adaptation 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 training rollouts and retaining task-wise experts 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):

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

where τ_k is the trajectory of episode k, $J(\tau_k)$ is its return, and $J_{\max}(g)$ is the maximum achievable return for task g. Later episodes receive larger weights, rewarding sustained improvement.

3.2. Learnable Adaptation Policies for Language Agents

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 adaptation 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); 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 adaptation 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, making adaptation lightweight, and casts the problem as a natural-language generation task that can itself be improved through meta-training.

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 adaptation 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 fully specifies the adaptation 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 itself—rather than hand-crafting it or relying on fixed heuristics, we optimize it through meta-training (§3.3).

3.3. Evolutionary Meta-Training

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

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

Algorithm 1 RUNTTL(ϕ, g): Runs a TTL session consisting of K episodes

Require: Meta-prompt ϕ; task g; initial actor prompt ρ_1; episode budget K
Require: ADAPT$(\phi, \mathcal{H})$: meta-agent call that generates an updated actor prompt given history $\mathcal{H}$ (Eq. 3)
for $k = 1, \ldots, K$ **do**
  $\tau_k \leftarrow$ RUNACTOR(g, ρ_k) # execute episode k under current actor prompt
  if $k < K$ **then**
    $\rho_{k+1} \leftarrow$ ADAPT$(\phi, \{\tau_1, \ldots, \tau_k\})$ # meta-agent rewrites actor prompt for next episode
  end if
end for
return $\xi_g^\phi \leftarrow (\tau_1, \ldots, \tau_K)$

Algorithm 2 Evolutionary Meta-Training of the Adaptation Policy

Require: Expert pool $\mathcal{P}$ from seed meta-prompt ϕ_0; training tasks $\mathcal{D}_{\text{train}}$; validation tasks $\mathcal{D}_{\text{val}}$; budget T
for $t = 1, \ldots, T$ **do**
  Sample $\phi_{\text{parent}} \sim \mathcal{P}$ and $g \sim \mathcal{D}_{\text{train}}$
  $\xi_{\text{parent}} \leftarrow$ RUNTTL$(\phi_{\text{parent}}, g)$ # run TTL session with the parent prompt
  $\phi_{\text{candidate}} \leftarrow$ PROPOSE$(\phi_{\text{parent}}, \xi_{\text{parent}})$ # reflect on parent run; propose candidate
  $\xi_{\text{candidate}} \leftarrow$ RUNTTL$(\phi_{\text{candidate}}, g)$ # local validation on the same task
  $s_{\text{parent}} \leftarrow$ W-AUC(ξ_{parent})
  $s_{\text{candidate}} \leftarrow$ W-AUC$(\xi_{\text{candidate}})$
  if $s_{\text{candidate}} \leq s_{\text{parent}}$ **then**
    continue # discard if there is no local improvement
  end if
  for $h \in \mathcal{D}_{\text{val}}$ **do**
    $\xi_h \leftarrow$ RUNTTL$(\phi_{\text{candidate}}, h)$ # global validation on all validation tasks
    $s_h \leftarrow$ W-AUC(ξ_h)
    if $s_h > \mathcal{P}[h]$.score **then**
      $\mathcal{P}[h] \leftarrow (\phi_{\text{candidate}}, s_h)$ # Expert Pool update; see §3.3
    end if
  end for
end for
$\phi^* \leftarrow$ SELECTEXPERT$(\mathcal{P})$ # select the top expert for deployment
return ϕ^*

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

Algorithm 1 defines the RUNTTL inner loop and Algorithm 2 shows the outer meta-training loop. The expert pool is initialized by scoring the seed prompt ϕ_0 on each validation task, and SCORE(ξ) denotes W-AUC in our experiments.

Proposal and Local Validation. Each iteration samples a parent meta-prompt from the current expert pool and a training task from $\mathcal{D}_{\text{train}}$, and runs a TTL session on that task with the meta-agent governed by the sampled meta-

Table 1. **Jericho ID Results.** Comparison of single-agent baselines and actor+meta-agent methods on meta-training games. Single-agent baselines are shown at the top for reference; within each meta-agent block, **META-TTL** improves overall performance over the Naive variant.

Architecture	Method	Avg. Score ↑				W-AUC ↑			
		Detective	Zork 1	Temple	Avg.	Detective	Zork 1	Temple	Avg.
Single-Agent Baselines	Static	91.7	38.5	5.0	45.1	0.24	0.11	0.14	0.16
	Reflexion	100.7	41.4	4.3	48.8	0.28	0.12	0.12	0.17
	Memory Agent	103.7	42.7	5.0	50.5	0.28	0.13	0.14	0.18
Gemini 3 Flash (as Meta-Agent)	Naive	104.9	40.8	4.0	49.9	0.29	0.12	0.11	0.17
	META-TTL	**115.6**	**43.0**	**7.2**	**55.3**	**0.33**	**0.13**	**0.22**	**0.23**
GLM-5 (as Meta-Agent)	Naive	122.7	39.2	3.7	55.2	0.37	0.11	0.10	0.19
	META-TTL	**224.0**	**47.3**	**7.4**	**92.9**	**0.68**	**0.14**	**0.22**	**0.35**
GPT-5 (as Meta-Agent)	Naive	107.9	38.7	4.7	50.4	0.31	0.11	0.13	0.18
	META-TTL	**270.5**	**53.7**	**8.1**	**110.8**	**0.82**	**0.16**	**0.24**	**0.41**

Table 2. **Jericho OOD Results.** Zero-shot generalization on held-out games, with single-agent baselines shown at the top for reference. **META-TTL** generally improves performance across all three meta-agent backbones.

Architecture	Method	Avg. Score ↑				W-AUC ↑			
		Balances	Library	Zork 3	Avg.	Balances	Library	Zork 3	Avg.
Single-Agent Baselines	Static	7.0	4.0	1.8	4.3	0.13	0.13	0.24	0.17
	Reflexion	7.4	8.9	2.0	6.1	0.14	0.32	0.28	0.25
	Memory Agent	7.7	8.0	1.8	5.8	0.15	0.29	0.24	0.23
Gemini 3 Flash (as Meta-Agent)	Naive	7.8	8.3	1.6	5.9	0.16	0.30	0.22	0.23
	META-TTL	**8.7**	**9.7**	**2.0**	**6.8**	**0.18**	**0.36**	**0.28**	**0.27**
GLM-5 (as Meta-Agent)	Naive	7.7	8.9	1.7	6.1	0.15	0.31	0.24	0.23
	META-TTL	**9.9**	**9.3**	**1.9**	**7.0**	**0.21**	**0.32**	**0.26**	**0.26**
GPT-5 (as Meta-Agent)	Naive	9.4	8.9	1.4	6.6	0.20	0.30	0.19	0.23
	META-TTL	**11.2**	**10.0**	**1.8**	**7.7**	**0.25**	**0.35**	**0.24**	**0.28**

prompt (Algorithm 2, lines 2–3). The proposer LLM then reads the resulting session and proposes a revised candidate (line 4). This candidate is re-evaluated on the same task; only candidates that improve W-AUC on that task proceed to global validation (lines 5–8).

Expert Pool. Similar to the per-task candidate tracking in GEPA (Agrawal et al., 2026), the expert pool stores the best meta-prompt found so far for each task in $\mathcal{D}_{\text{val}}$. A candidate that passes the local validation is evaluated on all validation tasks and replaces the current expert for every task on which it achieves a new best score (Algorithm 2, lines 10–16).

Expert Selection. After the meta-training budget is exhausted, the expert pool contains a set of specialized meta-prompts. We select a single meta-prompt ϕ^* for deployment by choosing the expert with the highest average validation score. When per-task reward scales differ substantially across the benchmark, we normalize via per-task z-scores before averaging to prevent easy-to-improve tasks from dominating the selection (details in Appendix A).

Evaluation. At test time, ϕ^* is frozen and deployed on tasks from the held-out set $\mathcal{D}_{\text{test}}$. The meta-agent updates the actor's system prompt between episodes exactly as during training, but ϕ^* itself is no longer modified.

4. Experiments

We evaluate META-TTL on two benchmarks with in-distribution (ID) and out-of-distribution (OOD) splits and study three research questions (RQs):

- **RQ1:** Does the meta-learned adaptation policy yield stronger test-time improvement than hand-crafted or unoptimized adaptation?
- **RQ2:** Does the learned adaptation policy generalize to out-of-distribution tasks?
- **RQ3:** What adaptation strategies emerge from evolutionary meta-training, and what mechanisms underlie their effectiveness?

Table 3. **WebArena-Lite ID Results.** Comparison of single-agent baselines and actor+meta-agent methods on in-distribution domains. Within each meta-agent block, **META-TTL** improves average performance across all meta-agent models.

Architecture	Method	Avg. Score ↑				W-AUC ↑			
		GitLab	Map	Shopping	Avg.	GitLab	Map	Shopping	Avg.
Single-Agent Baselines	Static	0.60	0.48	0.70	0.59	0.60	0.47	0.70	0.59
	Reflexion	0.60	0.44	0.68	0.57	0.60	0.46	0.69	0.58
	Memory Agent	0.60	0.46	0.70	0.59	0.60	0.45	0.70	0.58
Gemini 3 Flash (as Meta-Agent)	Naive	0.60	0.42	0.70	0.57	0.60	0.47	0.70	0.59
	META-TTL	0.60	**0.62**	**0.74**	**0.65**	0.60	**0.66**	**0.74**	**0.67**
GLM-5 (as Meta-Agent)	Naive	0.54	0.42	0.70	0.55	0.52	0.42	0.70	0.55
	META-TTL	**0.58**	**0.60**	**0.72**	**0.63**	**0.59**	**0.60**	**0.73**	**0.64**
GPT-5 (as Meta-Agent)	Naive	0.58	**0.48**	0.66	0.57	0.57	0.47	0.66	0.57
	META-TTL	0.58	0.46	**0.74**	**0.59**	**0.59**	0.47	**0.76**	**0.61**

Table 4. **WebArena-Lite OOD Results.** Zero-shot generalization on out-of-distribution domains, with single-agent baselines shown at the top for reference. **META-TTL** improves OOD average performance for all meta-model backbones.

Architecture	Method	Avg. Score ↑			W-AUC ↑		
		Reddit	Shopping Admin	Avg.	Reddit	Shopping Admin	Avg.
Single-Agent Baselines	Static	0.16	0.46	0.31	0.16	0.47	0.32
	Reflexion	0.16	0.48	0.32	0.16	0.48	0.32
	Memory Agent	0.16	0.47	0.32	0.16	0.46	0.31
Gemini 3 Flash (as Meta-Agent)	Naive	0.14	0.45	0.30	0.15	0.46	0.31
	META-TTL	**0.18**	**0.47**	**0.33**	**0.19**	**0.49**	**0.34**
GLM-5 (as Meta-Agent)	Naive	0.17	0.53	0.35	0.17	0.53	0.35
	META-TTL	0.17	**0.56**	**0.37**	0.17	**0.58**	**0.38**
GPT-5 (as Meta-Agent)	Naive	0.16	0.45	0.30	0.16	0.44	0.30
	META-TTL	0.16	**0.48**	**0.32**	0.16	**0.49**	**0.33**

4.1. Experimental Setup

Benchmarks. We evaluate META-TTL on two benchmarks: **Jericho** (Hausknecht et al., 2020), a suite of interactive fiction games, and **WebArena-Lite** (Zhou et al., 2024), a web-navigation benchmark with binary rewards. For Jericho, we use three ID games (Detective, Zork 1, Temple) for meta-training and ID evaluation, and three OOD games (Balances, Library, Zork 3) for zero-shot generalization. For WebArena-Lite, we split five website domains into ID (Shopping, GitLab, Map) and OOD (Reddit, Shopping Admin), with the ID domains further divided into training, validation, and evaluation subsets. Each Jericho session consists of 6 episodes, while each WebArena-Lite session consists of 5 episodes.

Models and Baselines. In all settings, the actor is a frozen **Gemini 3 Flash**, serving as the actor policy π. We evaluate three meta-agent backbones for the adaptation policy ϕ—**Gemini 3 Flash**, **GLM-5**, and **GPT-5**—each with its own independently meta-trained ϕ^*. We compare META-TTL with four baselines: **Static** (no adaptation), **Reflexion** (Shinn et al., 2023), **Memory Agent** (He et al., 2025), and a **Naive** meta-agent that uses the same actor–meta-agent architecture but without meta-trained adaptation policy. We report **Average Score** and **W-AUC**. Across all training runs, we use GPT-5.2 as the reflection LM in the outer loop.

4.2. Main Results (RQ1)

META-TTL consistently improves W-AUC relative to hand-crafted and unoptimized adaptation on both benchmarks. On Jericho ID games (Table 1), the gains are large for every meta-agent backbone, reaching 0.18→0.41 with GPT-5. On WebArena-Lite ID domains (Table 3), gains are smaller but still positive for all three backbones, up to +0.09 W-AUC with GLM-5.

One plausible reason for the gap between benchmarks is the difference in reward granularity. Jericho's dense, per-

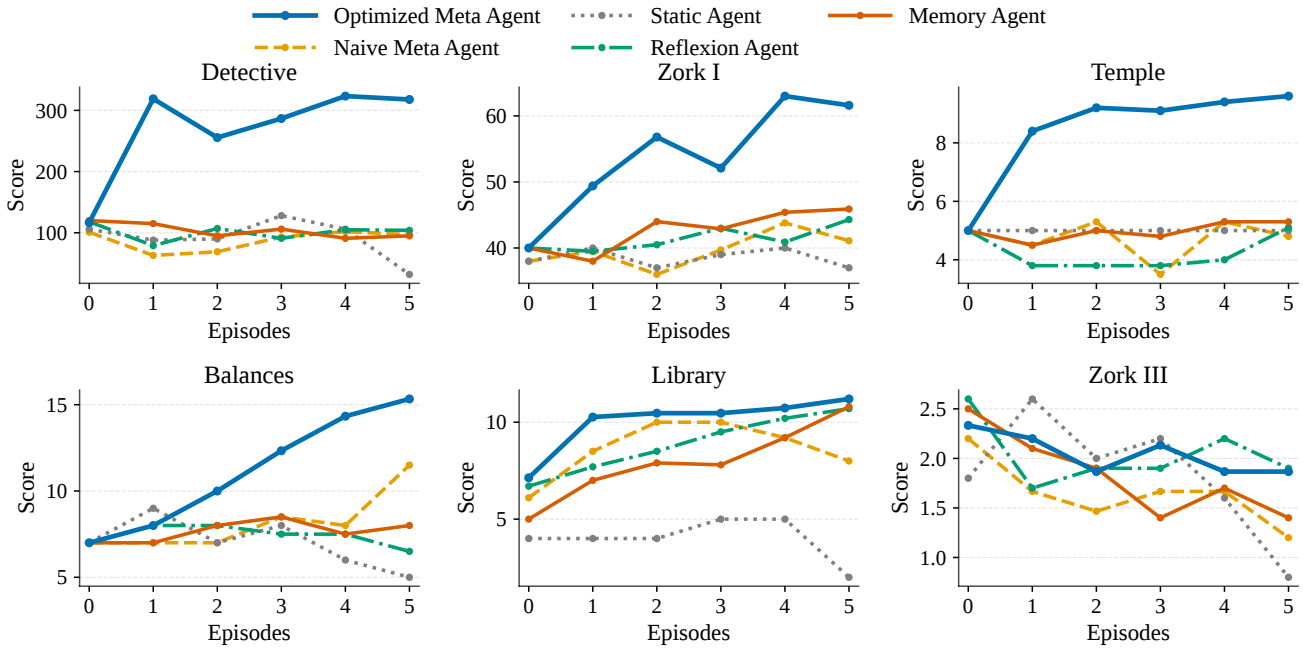


Figure 3. Per-episode score trajectories on the six Jericho evaluation games. META-TTL exhibits clearer upward trends across episodes than the baselines, supporting W-AUC as a metric of sustained test-time improvement.

action rewards provide a fine-grained optimization signal for meta-training: even a small improvement in the candidate adaptation policy is likely to be reflected in the session score, making it easier for evolutionary search to identify and retain better candidates. On WebArena-Lite, the binary completion signal yields session trajectories that are predominantly all-zero or all-one, offering a much coarser search landscape for the outer loop.

Within Jericho, META-TTL improves W-AUC on all six games (Tables 1–2). The hardest case is Zork 3, where all methods show declining scores across episodes, suggesting that effective test-time learning is difficult to achieve on this game. Even in this case, the optimized adaptation policy yields a higher W-AUC than the Naive baseline (e.g., 0.19→0.24 with GPT-5).

Beyond aggregate W-AUC, the per-episode score trajectories offer a more direct view of how learning unfolds across episodes. Figure 3 shows that META-TTL produces notably more stable learning curves on all six Jericho games. This holds even on Zork 3, where absolute improvement is limited but the trajectory yielded by META-TTL is notably more stable. On Detective, there is a sharper contrast: the Naive meta-agent's first feedback *degrades* the actor's score (114→89), whereas META-TTL yields a 2.7× improvement on the same transition (117→319; Appendix E).

4.3. Analysis of OOD Generalization (RQ2)

The learned adaptation policy generalizes to out-of-distribution tasks on both benchmarks (Tables 2 and 4). On Jericho, META-TTL improves W-AUC on all three held-out games across all meta-agent backbones; with GPT-5, the average rises from 0.23 to 0.28. This includes Zork 3, the most challenging of the three held-out games, where all backbones still record positive gains—suggesting that the learned adaptation policy is not overfit to the training distribution and generalizes well to unseen tasks.

On WebArena-Lite, OOD gains concentrate in Shopping Admin, which improves consistently across all three backbones (+0.03 to +0.05 W-AUC); Shopping Admin shares interface and task structure with the ID Shopping domain. Reddit, by contrast, is structurally unlike any training domain and shows more limited transfer: only Gemini 3 Flash records a meaningful gain (+0.04 W-AUC), while GLM-5 and GPT-5 remain flat. The overall OOD average nonetheless improves for all three backbones, driven primarily by Shopping Admin.

4.4. Analysis of Emergent Adaptation Policies (RQ3)

The optimized ϕ^* consists of task-agnostic adaptation strategies and environment-specific domain knowledge (Appendix C). The adaptation strategies include general strategies such as how the meta-agent should perform **credit assignment** over episode outcomes, **extract and consolidate knowledge** from observed trajectories, and **balance**

exploitation of known strategies with disciplined exploration. These strategies appear consistently across both Jericho and WebArena-Lite, suggesting they reflect general principles of effective adaptation rather than surface patterns specific to either benchmark.

Example: emergent adaptation strategies for Jericho. The full content is in Appendix B.

```
Content Requirements. Under the six
headings, the meta-agent MUST:
• Call out: (a) what scored points,
  (b) what caused death, (c) what
  wasted turns, (d) what blocked
  progress.
• Extract durable facts: map links,
  required triggers, working command
  syntax, non-working verbs.
• Provide a 15--25 command opening
  script reproducing known scoring
  actions before attempting new
  objectives.
• Exploration: ONE new branch at a
  time under SAVE/RESTORE. After two
  failures, switch strategy.
```

Meta-training progressively separated adaptation strategies from domain knowledge. The optimization trajectory (Appendix D) shows that early iterations hard-code environment-specific knowledge into the meta-prompt, achieving strong performance on individual training games but transferring poorly to others. As meta-training continues, cross-task validation in the expert pool drives the adaptation policy to factor into two components: task-agnostic adaptation strategies (as described above) and conditional fact banks that activate domain knowledge only when the current episode log confirms the relevant environment. This factorization emerged from the evolutionary process as a natural consequence of satisfying all validation tasks simultaneously.

The optimized meta-prompt ϕ^* can be viewed abstractly as a natural-language learning algorithm. The adaptation strategies it encodes are general adaptation procedures broadly applicable to agents learning from experience—expressed in natural language and executed by a language model rather than encoded as model weights, making the learned adaptation policy both interpretable and transferable across model backbones. The fact that meta-training discovers these adaptation rules from task performance alone, without hand-engineering their form, suggests that effective adaptation is itself a learnable procedure. Detailed case studies are provided in Appendix E.

5. Conclusion

This paper introduces **META-TTL**, a bi-level framework that learns adaptation policies for test-time learning in language agents through evolutionary meta-training across tasks. Across Jericho and WebArena-Lite, the learned adaptation policy consistently outperforms hand-crafted and unoptimized alternatives and transfers to out-of-distribution environments, while the meta-training process autonomously discovers interpretable adaptation strategies such as explicit credit assignment and disciplined exploration management. These results suggest that how an agent adapts from experience is itself a learnable component, and that progress in language-agent test-time learning may benefit from optimizing the adaptation procedure itself.

Implication Statement

This paper studies how language agents can learn to adapt at test time from experience, without weight updates. Because the learned adaptation policy is instanstiated in the form of natural language, it is human-readable and can be audited by humans to understand how the agent revises its own behavior across episodes, which we view as a useful property for transparency relative to opaque, weight-level adaptation. At the same time, more effective test-time learning increases the capability of autonomous agents acting on real-world surfaces such as the web, and can amplify existing misuse vectors (e.g., scaled scraping, automated account actions, or content production); the same evolutionary procedure that discovers useful adaptation strategies could in principle discover undesirable ones if the meta-training task distribution incentivizes them. We therefore view careful curation of meta-training tasks, sandboxed evaluation environments, and continued human oversight of deployed adaptation policies as essential complements to the technical progress reported here, and conduct our own experiments entirely on offline benchmarks (Jericho and WebArena-Lite) under such conditions.

References

Acikgoz, E. C., Qian, C., Ji, H., Hakkani-Tür, D., and Tur, G. Self-improving llm agents at test-time, 2025. URL https://arxiv.org/abs/2510.07841.

Agrawal, L. A., Tan, S., Soylu, D., Ziems, N., Khare, R., Opsahl-Ong, K., Singhvi, A., Shandilya, H., Ryan, M. J., Jiang, M., Potts, C., Sen, K., Dimakis, A., Stoica, I., Klein, D., Zaharia, M., and Khattab, O. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Akyürek, E., Damani, M., Zweiger, A., Qiu, L., Guo, H., Pari, J., Kim, Y., and Andreas, J. 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.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Chen, A., Liu, Z., Zhang, J., Prabhakar, A., Liu, Z., Heinecke, S., Savarese, S., Zhong, V., and Xiong, C. Grounded test-time adaptation for LLM agents. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=OH4PE0TDo0.

Chen, X., Xu, C., Wang, Y., Liu, B., Yao, Z., and He, Y. Learning to self-evolve, 2026b. URL https://arxiv.org/abs/2603.18620.

Chhikara, P., Khant, D., Aryan, S., Singh, T., and Yadav, D. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Dherin, B., Munn, M., Mazzawi, H., Wunder, M., and Gonzalvo, J. Learning without training: The implicit dynamics of in-context learning, 2025. URL https://arxiv.org/abs/2507.16003.

Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Liu, T., Chang, B., Sun, X., Li, L., and Sui, Z. A survey on in-context learning, 2024. URL https://arxiv.org/abs/2301.00234.

Fang, J., Peng, Y., Zhang, X., Wang, Y., Yi, X., Zhang, G., Xu, Y., Wu, B., Liu, S., Li, Z., Ren, Z., Aletras, N., Wang, X., Zhou, H., and Meng, Z. 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.

Fernando, C., Banarse, D., Michalewski, H., Osindero, S., and Rocktäschel, T. Promptbreeder: Self-referential self-improvement via prompt evolution, 2023. URL https://arxiv.org/abs/2309.16797.

Gao, H., Geng, J., Hua, W., Hu, M., Juan, X., Liu, H., Liu, S., Qiu, J., Qi, X., Wu, Y., Wang, H., Xiao, H., Zhou, Y., Zhang, S., Zhang, J., Xiang, J., Fang, Y., Zhao, Q., Liu, D., Ren, Q., Qian, C., Wang, Z., Hu, M., Wang, H., Wu, Q., Ji, H., and Wang, M. 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.

Guo, Q., Wang, R., Guo, J., Li, B., Song, K., Tan, X., Liu, G., Bian, J., and Yang, Y. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers, 2025. URL https://arxiv.org/abs/2309.08532.

Gupta, P., Kirtania, S., Singha, A., Gulwani, S., Radhakrishna, A., Shi, S., and Soares, G. Metareflection: Learning instructions for language agents using past reflections, 2024. URL https://arxiv.org/abs/2405.13009.

Hausknecht, M., Ammanabrolu, P., Côté, M.-A., and Yuan, X. Interactive fiction games: A colossal adventure, 2020. URL https://arxiv.org/abs/1909.05398.

He, Y., Liu, J., Liu, Y., Li, Y., Cao, T., Hu, Z., Xu, X., and Hooi, B. Evotest: Evolutionary test-time learning for self-improving agentic systems, 2025. URL https://arxiv.org/abs/2510.13220.

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

Jiang, P., Lin, J., Shi, Z., Wang, Z., He, L., Wu, Y., Zhong, M., Song, P., Zhang, Q., Wang, H., Xu, X., Xu, H., Han, P., Zhang, D., Sun, J., Yang, C., Qian, K., Wang, T., Hu, C., Li, M., Li, Q., Peng, H., Wang, S., Shang, J., Zhang, C., You, J., Liu, L., Lu, P., Zhang, Y., Ji, H., Choi, Y., Song, D., Sun, J., and Han, J. Adaptation of agentic ai: A survey of post-training, memory, and skills, 2026a. URL https://arxiv.org/abs/2512.16301.

Jiang, Y., Jiang, L., Teney, D., Moor, M., and Brbic, M. Meta-rl induces exploration in language agents, 2026b. URL https://arxiv.org/abs/2512.16848.

Kumar, A., Zhuang, V., Agarwal, R., Su, Y., Co-Reyes, J. D., Singh, A., Baumli, K., Iqbal, S., Bishop, C., Roelofs, R., et al. Training language models to self-correct via reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2024.

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

Liu, S., Agarwal, S., Maheswaran, M., Cemri, M., Li, Z., Mang, Q., Naren, A., Boneh, E., Cheng, A., Pan, M. Z., Du, A., Keutzer, K., Cheung, A., Dimakis, A. G., Sen, K., Zaharia, M., and Stoica, I. Evox: Meta-evolution for automated discovery, 2026. URL https://arxiv.org/abs/2602.23413.

Liu, T. and van der Schaar, M. 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.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

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

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., and Balog, M. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., and Gonzalez, J. E. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Suzgun, M., Yuksekgonul, M., Bianchi, F., Jurafsky, D., and Zou, J. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025. URL https://arxiv.org/abs/2504.07952.

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

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Wei, T., Sachdeva, N., Coleman, B., He, Z., Bei, Y., Ning, X., Ai, M., Li, Y., He, J., Chi, E. H., Wang, C., Chen, S., Pereira, F., Kang, W.-C., and Cheng, D. Z. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory, 2025. URL https://arxiv.org/abs/2511.20857.

Wu, C.-K., Tam, Z. R., Lin, C.-Y., Chen, Y.-N., and yi Lee, H. Streambench: Towards benchmarking continuous improvement of language agents, 2024. URL https://arxiv.org/abs/2406.08747.

Xiao, T., Yuan, Y., Ivison, H., Zhu, H., Brahman, F., Lambert, N., Dasigi, P., Smith, N. A., and Hajishirzi, H. Meta-reinforcement learning with self-reflection for agentic search, 2026. URL https://arxiv.org/abs/2603.11327.

Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., and Zhang, Y. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. Large language models as optimizers, 2024. URL https://arxiv.org/abs/2309.03409.

Ye, H., Wang, J., Cao, Z., Berto, F., Hua, C., Kim, H., Park, J., and Song, G. Reevo: Large language models as hyper-heuristics with reflective evolution, 2024. URL https://arxiv.org/abs/2402.01145.

Ye, T., Dong, L., Dong, Q., Wu, X., Huang, S., and Wei, F. Online experiential learning for language models, 2026. URL https://arxiv.org/abs/2603.16856.

Yuksekgonul, M., Bianchi, F., Boen, J., Liu, S., Huang, Z., Guestrin, C., and Zou, J. Textgrad: Automatic "differentiation" via text, 2024. URL https://arxiv.org/abs/2406.07496.

Yuksekgonul, M., Koceja, D., Li, X., Bianchi, F., McCaleb, J., Wang, X., Kautz, J., Choi, Y., Zou, J., Guestrin, C., and Sun, Y. Learning to discover at test time, 2026. URL https://arxiv.org/abs/2601.16175.

Zelikman, E., Wu, Y., Mu, J., and Goodman, N. STar: Bootstrapping reasoning with reasoning. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_3ELRdg2sgI.

Zhang, K., Chen, X., Liu, B., Xue, T., Liao, Z., Liu, Z., Wang, X., Ning, Y., Chen, Z., Fu, X., Xie, J., Sun, Y., Gou, B., Qi, Q., Meng, Z., Yang, J., Zhang, N., Li, X., Shah, A., Huynh, D., Li, H., Yang, Z., Cao, S., Jang, L., Zhou, S., Zhu, J., Sun, H., Weston, J., Su, Y., and Wu, Y. Agent learning via early experience, 2025. URL https://arxiv.org/abs/2510.08558.

Zhang, Q., Hu, C., Upasani, S., Ma, B., Hong, F., Kamanuru, V., Rainton, J., Wu, C., Ji, M., Li, H., Thakker, U., Zou, J., and Olukotun, K. Agentic context engineering: Evolving contexts for self-improving language models, 2026. URL https://arxiv.org/abs/2510.04618.

Zhou, H., Chen, Y., Guo, S., Yan, X., Lee, K. H., Wang, Z., Lee, K. Y., Zhang, G., Shao, K., Yang, L., and Wang, J. Memento: Fine-tuning llm agents without fine-tuning llms, 2025. URL https://arxiv.org/abs/2508.16153.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., and Neubig, G. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., and Ba, J. Large language models are human-level prompt engineers, 2023. URL https://arxiv.org/abs/2211.01910.

Zuo, Y., Zhang, K., Sheng, L., Qu, S., Cui, G., Zhu, X., Li, H., Zhang, Y., Long, X., Hua, E., Qi, B., Sun, Y., Ma, Z., Yuan, L., Ding, N., and Zhou, B. TTRL: Test-time reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=VuVhgEiu20.

Zweiger, A., Pari, J., Guo, H., Kim, Y., and Agrawal, P. Self-adapting language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=JsNUE84Hxi.

A. Expert Selection and Score Normalization

For **WebArena-Lite**, every task yields a binary completion signal, so the reward scales are directly comparable across tasks. We therefore select the candidate with the highest raw average success rate across validation tasks.

For **Jericho**, score normalization requires more care. Although W-AUC already divides by the maximum attainable score $J_{\max}(g)$ (Eq. 1), games inherently differ in how easy they are to improve on. For instance, Detective is substantially easier to improve on than Temple or Zork 1, so selecting by raw W-AUC average can favor candidates overfitted to a single easy game, leading to a less generalizable expert.

To correct for this, we apply a post-hoc per-game z-score normalization over the full set of candidates evaluated during meta-training. For each game g, we compute the mean μ_g and standard deviation σ_g of W-AUC scores across all candidates that reached the global validation stage, normalize each candidate's score as $z_{i,g} = (s_{i,g} - \mu_g)/\sigma_g$, and select the candidate with the highest average z-score across games.

Illustrative example. In an example meta-training run, the raw-average winner is Candidate P5 (average W-AUC 0.371), while the z-score winner is Candidate P11 (average W-AUC 0.348). Table 5 shows why. P5's raw-score lead of $+0.107$ on Detective looks large, but Detective has a high σ_g (0.108), so this gap amounts to only $+1.00z$. By contrast, P11's advantage of $+0.030$ on Zork 1 is small in raw score but Zork 1 is much harder to improve on ($\sigma_g = 0.013$), making it worth $+2.31z$. P11 is also stronger on Temple ($+0.38z$). Overall, P11 achieves a substantially higher average z-score ($+0.96$) than P5 ($+0.40$), and is selected as the more *uniformly* strong candidate.

Table 5. Expert selection example from a representative meta-training run. P5 wins on raw W-AUC average, but P11 wins after per-game z-score normalization. The per-game σ_g row shows why: Detective is high-variance, so P5's large raw lead there carries less weight once normalized.

Candidate	Raw W-AUC			Z-Score			Avg. z
	Detective	Zork 1	Temple	Detective	Zork 1	Temple	
Per-game μ_g	0.554	0.145	0.197	—	—	—	—
Per-game σ_g	0.108	0.013	0.019	—	—	—	—
P11 (z-score winner)	0.675	**0.161**	**0.207**	$+1.13$	$\mathbf{+1.27}$	$\mathbf{+0.48}$	$\mathbf{+0.96}$
P5 (raw-avg winner)	**0.783**	0.131	0.199	$\mathbf{+2.12}$	-1.04	$+0.10$	$+0.40$

B. Representative Optimized Meta-Prompts

We show two representative optimized meta-prompts: the GPT-5 prompt used for Jericho and the Gemini 3 Flash prompt used for WebArena-Lite. On Jericho, GPT-5's prompt is the clearest instance of the the emergent properties analyzed in Appendix C. On WebArena-Lite, the GLM-5 prompt better reflects the benchmark-level adaptation policy than the more task-specific GPT-5 and Gemini variants. Structural skeletons are shown below; full prompts including complete fact banks are available in the supplementary material.

B.1. Jericho GPT-5 Prompt

We reproduce the structural skeleton below, abbreviating the per-game fact banks for space. The full prompt is available in the supplementary material.

ROLE
You are a META-AGENT for parser-based interactive fiction (IF). You read an Actor Agent's MOST RECENT episode transcript and produce a "guiding prompt" for the NEXT episode.

NON-NEGOTIABLE CONSTRAINTS

- The next episode ALWAYS restarts from the beginning.
- Tight move budget (∼50 turns). Optimize points-per-turn.
- Never invent rooms, items, NPCs, or scoring rules.

- At most ONE EXPERIMENT per episode, under SAVE/RESTORE.

CRITICAL ADAPTATION RULE (GAME IDENTIFICATION FIRST) ⑥

1. Identify the actual game from the newest log (title, rooms, items, NPCs).
2. Use a Fact Bank ONLY if the log clearly matches that game.
3. If no match, ignore all Fact Banks and build facts strictly from the log.

REQUIRED OUTPUT FORMAT (STRICT) ①
Return EXACTLY these six headings:

1. What happened / diagnosis
2. Game facts to remember ③
3. Next-episode priorities
4. Recommended route (with save points)
5. Command script (first ∼15-25 moves) ⑤
6. Parser tips specific to this game

CONTENT REQUIREMENTS
Under the six headings, you MUST:

- Call out: (a) what scored points, (b) what caused death, (c) what wasted turns, (d) what blocked progress. ②
- Extract durable facts: map links, required triggers, working command syntax, non-working verbs. ③
- Provide a 15-25 command opening script reproducing known scoring actions before attempting new objectives. ⑤
- Exploration: ONE new branch at a time under SAVE/RESTORE. After two failures, switch strategy. ④

FACT BANKS (USE ONLY IF GAME MATCH IS CLEAR) ⑥

A) **Detective** -- [31 lines: room-by-room map, scoring actions, lethal traps, parser syntax for each stage from Chief's office through Holiday Inn win path]

B) **The Temple** -- [18 lines: key sequence including CLIMB CHARLES for iron key, vial/cat/slab puzzle, underground chemistry]

C) **Zork I** -- [10 lines: opening sequence, troll fight, dam cluster, maze warning, SAVE/RESTORE preference]

Output Format:
<think>your reasoning...</think>
<learn>Your derived useful feedback</learn>

B.2. WebArena-Lite Gemini 3 Flash Prompt

ROLE
You are a META-AGENT. Read multi-episode web-browsing trajectories and write a "Guiding Prompt" for the NEXT episode that corrects observed failure modes.

WHAT YOU MUST OUTPUT

- 3-7 numbered, action-oriented steps (a checklist/plan).
- UI-grounded: name exact clickable controls/fields using labels seen in the log.

```
  • Include at least one decision rule ("If X, do Z instead").

HOW TO BUILD THE GUIDING PROMPT

 1. Infer the user's actual task objective from the log + any prior feedback.

 2. Identify the precise point(s) of failure (wrong UI element, wrong entity,
    duplicate/blocked flow, didn't finalize, navigated away, etc.).

 3. Write steps that:

      • Navigate to the correct page deterministically (prefer sidebar navigation over
        searching).
      • Perform the minimal actions needed.
      • Add guardrails to prevent repeating prior mistakes.

CROSS-EPISODE PRIORITIES (common success patterns)

  • Always include a "finish" step aligned with the task (e.g., click the final submit
    button, or verify the existing artifact if creation is blocked).

  • If a form has multiple similar controls (e.g., Assignee vs Reviewer), name the
    correct one explicitly and warn against the distractor.

  • If the task might already be satisfied, instruct the agent to verify completion
    rather than recreate.

SITE-SPECIFIC KNOWLEDGE: GITLAB
[...  16 lines:  MR creation flow, duplicate MR handling, reviewer setting,
non-blocking warning handling ...]

DECISION RULE EXAMPLES

  • If expected button isn't present, backtrack to last stable page via canonical
    navigation path.

  • If creation is blocked by duplicate error, open the existing item and verify
    required fields instead.

## Output Format:
<think>reasoning...</think>
<learn>Your guiding prompt for the next episode</learn>
```

C. Emergent Properties of the Optimized Meta-Prompt

The optimized meta-prompt (ϕ^*), evolved through meta-training on three ID games, exhibits several qualitatively distinct features that were absent from the seed prompt and emerged entirely through the evolutionary optimization process:

1. ① **Mandatory structured output.** The meta-prompt specifies six required output sections: (1) diagnosis of what happened, (2) durable game facts, (3) next-episode priorities, (4) a recommended route with save points, (5) a concrete command script (first 15–25 moves), and (6) parser tips specific to the game. This structure forces the meta-agent to separate diagnosis, fact extraction, planning, and scripting rather than producing a monolithic narrative.
2. ② **Explicit credit assignment protocol.** The meta-prompt requires the meta-agent to itemize which actions scored points and how to reproduce them, which actions caused death or created threats, which actions wasted turns (dead ends, loops), and which actions blocked progress (locked doors, parser failures).
3. ③ **Grounded fact accumulation.** A "Game facts to remember" section must record map links, required triggers, working command syntax, and non-working verbs the parser rejected. Critically, the meta-prompt constrains these facts to be evidenced by the most recent episode log, preventing hallucination.
4. ④ **Exploration management.** The meta-prompt enforces a disciplined exploration policy: at most one new experiment per episode, always under a save/restore point, with an explicit fallback if two attempts at the same approach fail.
5. ⑤ **Concrete action scripts.** Rather than providing abstract strategic advice, the meta-prompt requires a 15–25 command opening script that reproduces known scoring actions quickly before attempting new objectives.

6. ⑥ **Conditional fact banks.** The meta-prompt includes game-specific knowledge (map layouts, scoring sequences, parser syntax, lethal traps) for each ID training game, activated only when the game identity is confirmed from the episode log. A "CRITICAL ADAPTATION RULE" ensures that irrelevant fact banks are ignored.

D. Meta-Training Optimization Trajectory

We trace the full optimization trajectory of the GPT-5 meta-agent on the three Jericho ID games (Detective, Zork 1, Temple), which ran for 26 iterations over approximately 27 hours. The seed meta-prompt ϕ_0 is a generic one-sentence instruction ("analyze the game trajectory and provide feedback"), achieving an aggregate validation W-AUC of 0.188. Of 26 proposals, 16 pass the local validation gate; of those, 6 achieve a new best aggregate score. Representative examples:

- **Iteration 1** (W-AUC: $0.188 \rightarrow 0.318$, $+69\%$): The proposer discovers that structured, game-aware feedback dramatically outperforms vague advice. The proposed prompt introduces turn-budget awareness ("tight move budget $\sim$50 turns"), episode-restart semantics, and game-specific context.
- **Iteration 5** ($0.318 \rightarrow 0.340$): Introduces the mandatory six-section output format (diagnosis, game facts, priorities, route, command script, parser tips), forcing the meta-agent to decompose its reflection into distinct subtasks.
- **Iteration 7** ($0.340 \rightarrow 0.344$): Adds a critical robustness fix for multi-game generalization (detailed below).
- **Iteration 14** ($0.344 \rightarrow 0.372$): Refines per-game fact banks with evidence-grounding constraints ("only restate facts supported by the most recent log").
- **Iteration 22** ($0.372 \rightarrow 0.407$): Integrates all prior improvements into a comprehensive prompt that becomes the final ϕ^*.

A concrete example: discovering the game-identification fix. The most instructive moment occurs at iteration 7. In iterations 1–4, the proposer, having seen high-scoring Detective trajectories, hardcodes "Detective by Matt Barringer, Inform 6" into the meta-prompt. This works well for Detective (per-game W-AUC: 0.621) but provides irrelevant guidance for Zork 1 (0.141) and Temple (0.193). By iteration 7, the proposer diagnoses this failure and introduces a game-identification rule: "Do NOT assume the game is always the one named anywhere else. Identify the actual game from the log. If the log's game differs from any stored facts, IGNORE unrelated facts." By iteration 22, this evolves into a refined "CRITICAL ADAPTATION RULE" with conditional fact banks. This transforms the meta-prompt from a single-game specialist into a game-agnostic framework, and illustrates that each evolutionary proposal is a semantically-informed mutation—the proposer diagnoses why the current candidate fails and generates a targeted fix, rather than perturbing randomly.

E. Case Studies

We compare the optimized meta-prompt (OPT) against the naive baseline (NAIVE) using the GPT-5 meta-agent with a Gemini 3 Flash actor on Jericho. Each case study highlights a different aspect of how the learned adaptation policy produces better feedback.

E.1. Detective (ID Game): Actionable vs. Generic Feedback

Both conditions start Episode 0 at comparable scores ($\sim$114), since the actor has no meta-agent guidance yet. The key divergence occurs at Episode 1, after the first feedback.

NAIVE produces generic interactive-fiction advice ("*Core loop per room: LOOK, then EXAMINE all notable objects; SEARCH room/containers...*"). This could apply to any game and does not leverage Episode 0 observations. The actor's score *drops* to 89.

OPT instead diagnoses the specific failure and prescribes a fix:

> "*Blocker: confronted the dazed man without the pistol and with wrong syntax. Wasted turns: skipped the pistol in Chief's west closet. Command script: GET PAPER / READ PAPER / WEST / GET PISTOL / ... / SHOOT DAZED MAN WITH PISTOL.*"

The actor's score jumps to 319—a 2.7$\times$ improvement in one feedback cycle. Over subsequent episodes, the meta-agent progressively tightens the route (diagnosing turn-budget bottlenecks, reordering scoring actions), reaching 340/360 by Episode 4. Under NAIVE, scores fluctuate between 89–131 with no upward trend.

E.2. Temple (ID Game): Diagnosing Non-Obvious Blockers

Temple (max 35 points) tests whether the meta-agent can identify unconventional actions. Under NAIVE, the actor never exceeds 5/35—it remains stuck in the study room because reaching the next area requires `CLIMB CHARLES` (climbing an NPC to retrieve a key), an action unlikely to be attempted without targeted guidance. Generic advice like "EXAMINE every object" does not surface this.

Under OPT, the Episode 1 feedback pinpoints the gap: "*You missed: CLIMB CHARLES for the iron key (+3), taking the vial, unlocking the oak door.*" The actor reaches 8–10 points by Episode 2, nearly doubling its score.

E.3. Transfer to OOD Games: Structural Rules Generalize

On Balances (OOD), the OPT meta-agent has never seen this game, yet its first feedback correctly connects the actor's observation to an available tool: "*the cedarwood box is locked; your spell book already lists rezrov*" $\rightarrow$ recommends `LEARN REZROV` then `CAST REZROV ON BOX`. The NAIVE meta-agent instead lists generic spells ("memorize FROTZ, YOMIN, REZROV, BOZBAR if available") without connecting any to the specific puzzle. The difference is that the optimized prompt's credit-assignment and blocker-identification format (Section 4 of the output template) forces the meta-agent to match each blocker to a concrete next action, even in an unseen game. The Naive meta-agent, lacking this structure, defaults to generic advice.