CHOOSE YOUR MOMENT: SELECTIVE EVOLUTION OF LLM AGENTS VIA LATENT REASONING DYNAMICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon LLM agents can continually improve by consolidating accumulated experience into their model parameters, yet indiscriminate updating is neither necessary nor always beneficial. Frequent updates incur substantial training costs and may overfit transient experience or interfere with previously acquired capabilities, whereas delayed updates can force agents to repeatedly reconstruct reasoning patterns that could have been internalized. This raises a key question for self-evolving agents: *when is model evolution actually worthwhile?* In this paper, we study this problem through the lens of latent reasoning dynamics and introduce LATCH, a selective evolution framework that encodes latent reasoning trajectories and extracts deep-to-shallow reasoning corrections. We further characterize whether these corrections are locally realizable by a shared parameter change and remain consistent across past queries, providing evidence of whether the accumulated reasoning patterns can be jointly internalized. These signals are then used to predict the finite-horizon advantage of updating immediately over waiting, supervised by paired update–wait rollouts that measure downstream task performance and capability retention. For evolution steps predicted to be worthwhile, we perform on-policy self-distillation, using a frozen deeper reasoning path to supervise the current shallow model on student-generated prefixes while using experience replay to mitigate degradation of prior capabilities. Experiments across long-horizon benchmarks demonstrate consistent improvements in cumulative task utility and capability retention with reduced update overhead.

1 INTRODUCTION

Large language models (LLMs) are increasingly deployed as agents that operate over extended interactions, creating opportunities to learn from experience accumulated (Zhang et al., 2026a; Shao et al., 2026; Wang et al., 2023). Consolidating this experience into the model can endow agents with capabilities that persist across subsequent tasks (Cao et al., 2026; Zheng et al., 2026). However, newly acquired experience does not necessarily justify an immediate update. A limited set of recent interactions may provide an incomplete basis for learning, while adapting to such limited evidence can compromise capabilities that remain useful elsewhere (Yu et al., 2026a; Qiu et al., 2025). Delaying an update allows additional evidence to accumulate, but may also postpone improvements that could benefit intervening tasks. This creates a fundamental tension between responsiveness and stability in long-horizon model evolution. For a continually operating agent, the central challenge is therefore to determine when accumulated experience is sufficiently valuable to justify evolution.

In self-evolving LLM agents, model updates are typically scheduled at fixed intervals (Zhang et al., 2026c;b; Weng et al., 2026) or triggered by observable signals such as error rate, uncertainty, and distribution changes (Xiao & Snoek, 2024; Chen et al., 2026; Gu et al., 2026). These strategies provide practical adaptation criteria, but only indirect evidence of whether evolving the model at a given moment will improve future behavior. Outcome-level signals characterize how the current policy handles recent experience, yet they do not reveal whether that experience contains reusable structure that can support future tasks (Xiong et al., 2024; Song et al., 2024). As a result, interaction histories with similar observed outcomes may differ substantially in their value for model evolution (Peng et al., 2025). Errors may arise from unrelated examples with little shared structure, whereas stable accuracy may conceal recurring reasoning patterns that the agent repeatedly reconstructs (Didolkar et al., 2025). Recent advances in adaptive and latent reasoning expose richer

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.

process-level information about how answers are formed (Chen & Li, 2024; Hao et al., 2025; Zhu et al., 2025). This information may reveal reusable reasoning structure in accumulated experience, but it remains insufficient to determine whether the model should evolve at the current moment. Update timing is therefore inherently comparative: evolving now must be evaluated against continuing with the current model while retaining the option to evolve after further experience (Piaseczny et al., 2025; ang Gao et al., 2026). This leads to our key question: *How can a long-running LLM agent determine whether evolving now will yield greater future benefit than waiting?*

To address these limitations, we introduce LATCH, a selective evolution framework built on **lat**ent reasoning traje**c**tories and deep-to-s**h**allow corrections. LATCH combines trajectories collected during normal operation with corrections revealed by deeper reasoning over past queries, and assesses whether these corrections can be jointly realized through a shared change to the reasoning adapter. This evidence is used to predict the finite-horizon advantage of evolving now over waiting, with supervision from paired update–wait continuations that capture both future task performance and capability retention. When evolution is favored, LATCH performs on-policy self-distillation (OPSD), using the deeper reasoning path to supervise the shallow policy while experience replay preserves previously acquired capabilities. Our contributions are summarized as follows:

- We formulate selective self-evolution as an update–wait decision, recognizing that current difficulty alone does not determine whether a parameter update is more valuable now than after further experience has accumulated.
- We introduce LATCH, which combines latent trajectory encoding, deep-to-shallow correction diagnostics, shared parameter realizability, and finite-horizon advantage prediction to trigger on-policy self-distillation at selected decision points.
- We conduct extensive experiments across various benchmarks, showing that LATCH improves cumulative task utility and capability retention via selective evolution.

2 Related Work

2.1 Continual Adaptation and Self-Evolving LLM Agents

LLM agents increasingly retain information from prior interactions and leverage accumulated experience to improve subsequent behavior (Xiong et al., 2026; Yu et al., 2026b; Yang et al., 2026a). Existing approaches broadly follow two directions. External memory, reflection, and retrieval mechanisms reuse prior trajectories without modifying model parameters (Dai et al., 2026; Yang et al., 2026a), whereas self-training, feedback-driven refinement, and test-time adaptation update the model itself, allowing experience to alter the underlying policy rather than only its context (Yang et al., 2026b; Wu et al., 2026a). Continual-learning methods further employ replay, regularization, and parameter-efficient adaptation to incorporate new knowledge while mitigating interference with previously acquired capabilities (Feng et al., 2026; Qiao & Mahdavi, 2026; Xiao et al., 2026). More recent work studies when such adaptation should occur, using periodic consolidation or selective triggers based on errors, uncertainty, novelty, and distribution change (Wu et al., 2026a).

Despite this progress, update timing still relies largely on predefined schedules or observable proxies of current difficulty. These signals can identify challenging experiences, but not whether adapting will improve future behavior. Since waiting allows additional evidence to accumulate, the decision is inherently sequential. Existing approaches therefore provide limited guidance on the downstream value of adapting now versus later.

2.2 Latent Reasoning and Reasoning Internalization

Recent work on LLM agents increasingly focuses on improving the reasoning process underlying their decisions rather than only final outputs. Chain-of-thought prompting, sampled deliberation, verifier-guided search, and inference-time scaling improve reasoning by allocating additional inference-time computation (Wei et al., 2022; Hosseini et al., 2024; Li & Cao, 2026; Snell et al., 2025). Latent-reasoning methods instead perform part of this computation through recurrent hidden-state transitions, continuous thought representations, or adaptive latent depth (Geiping et al., 2025; Koishekenov et al., 2025; Du et al., 2026). Reasoning distillation internalizes stronger reasoning behavior by transferring capabilities from deeper or more capable processes into efficient student

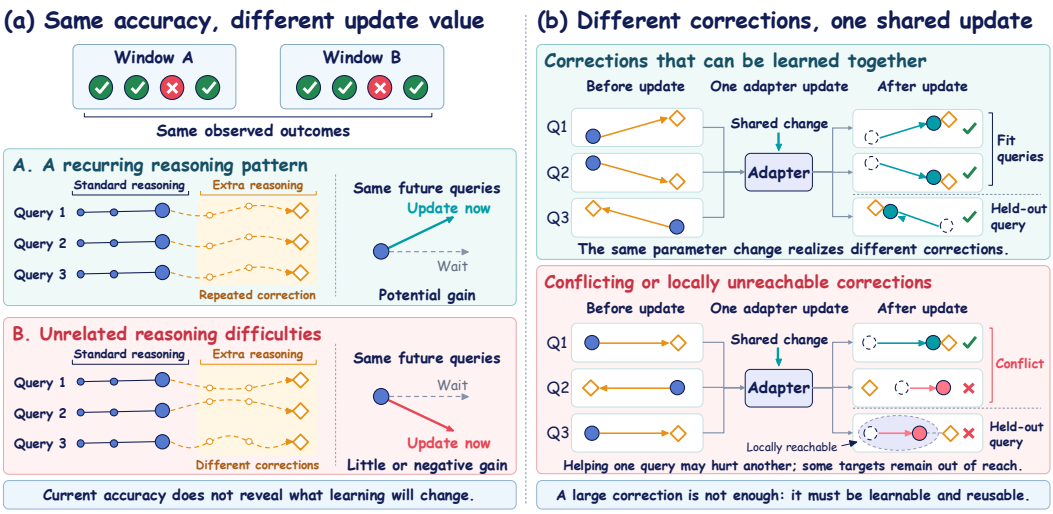


Figure 1: (a) Experience windows with identical observed accuracy can differ in their recurring reasoning patterns and potential benefits from updating. (b) A shared adapter update can realize different query-specific corrections, whereas conflicting corrections or locally unreachable targets limit their consolidation.

policies (Lee et al., 2025; Kang et al., 2025; Yang et al., 2025). On-policy variants supervise the student on states or prefixes induced by its own behavior, reducing the mismatch between training and deployment trajectories (Agarwal et al., 2024; Zhao et al., 2026).

Despite this progress, latent reasoning and reasoning distillation mainly focus on improving inference efficiency or transferring stronger reasoning behavior into a target model. They provide limited guidance on when such reasoning experience should be incorporated into the model during long-horizon agent operation. In particular, the presence of richer or more effective reasoning does not by itself indicate whether adapting the model at the current moment will improve future behavior.

3 Observations

We introduce two observations that motivate when accumulated experience warrants a parameter update, as illustrated in Figure 1, demonstrating why similar observable outcomes can have different update values and how distinct latent corrections may admit a shared parameter change.

Observation 1: Similar outcomes can imply different evolution advantages. Observable signals such as task outcomes, confidence, and recent error rates characterize current performance, but do not reveal how the policy would change after adaptation. Two experience windows with similar accuracy may therefore support different evolution decisions. One may exhibit a recurring reasoning pattern whose consolidation benefits subsequent queries, whereas the other may consist of unrelated difficulties with little reusable structure. Thus, comparable observable performance does not necessarily imply comparable evolution advantage.

Observation 2: Useful latent corrections admit a shared parameter update. Additional reasoning may reveal corrections that are not captured by the agent's usual reasoning process, but their presence alone does not justify model evolution. Useful corrections should be compatible across queries and admit a shared change in the trainable parameters. If multiple corrections can be approximated by the same local parameter change, they may be jointly internalized; conflicting or locally unreachable corrections instead limit reusable adaptation. Thus, useful corrections should be jointly compatible and locally realizable in parameter space.

4 Methodology

In this section, we present our framework in Figure 2. We formulate self-evolution as an update–wait decision in § 4.1, characterize latent reasoning dynamics and deep-to-shallow corrections in § 4.2,

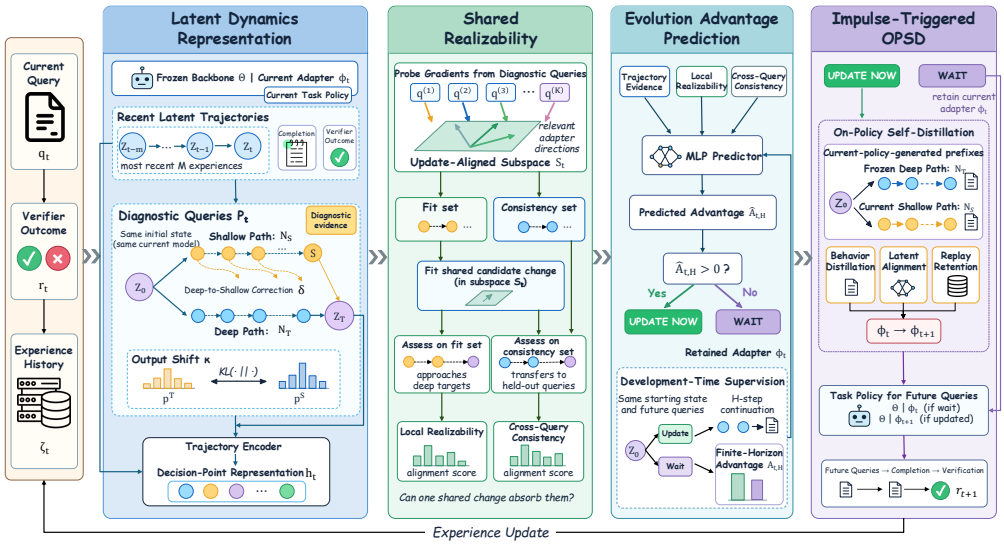


Figure 2: An overview of our proposed framework, LATCH. The diagram illustrates latent dynamics representation, shared realizability assessment, evolution advantage prediction, impulse-triggered on-policy self-distillation, and the feedback loop through subsequent agent experience.

predict finite-horizon evolution advantage from trajectory evidence and parameter realizability in § 4.3, and realize selected updates through impulse-triggered on-policy self-distillation in § 4.4.

4.1 PROBLEM FORMULATION

We consider a temporally ordered stream of queries $\{x_t\}_{t=1}^{T}$. Let θ_t denote the parameters of the agent at time step t. Given a query x_t, the agent performs multi-step latent reasoning and generates a completion $y_t \sim \pi_{\theta_t}(\cdot \mid x_t)$. A task-provided verifier $\mathcal{V}$ then evaluates the generated completion and returns a task outcome $r_t = \mathcal{V}(x_t, y_t, I_t)$, where I_t denotes the task-specific information required to determine correctness. For verifiable reasoning tasks, $\mathcal{V}$ may compare the generated answer with a reference solution, apply a deterministic validation rule, or use another task-defined correctness criterion. We denote by ζ_t the experience available up to time t, including historical queries, latent reasoning trajectories, generated completions, and verifier outcomes, which evolves with the task policy, as updates to θ_t affect the trajectories and outputs generated for subsequent queries. At each decision point, the evolution policy determines whether the agent should update. After the current query has been completed and evaluated, the policy observes (θ_t, ζ_t) and selects a meta-action $a_t \in \{0, 1\}$, where $a_t = 0$ denotes `wait` and $a_t = 1$ denotes `update`. Let $\mathcal{U}$ be a parameter-update operator that learns from experience available at the decision point. The agent evolves by

$$\theta_{t+1} = \theta_t + a_t[\mathcal{U}(\theta_t; \zeta_t) - \theta_t], \tag{1}$$

which determines whether the accumulated experience is sufficiently valuable to justify an update, accounting for expected benefits on future queries and effects on previously acquired capabilities. This formulation separates the evolution decision from the update operator: the evolution policy determines whether the accumulated experience justifies an update, while $\mathcal{U}$ specifies how the parameters evolve once an update is triggered.

4.2 LATENT DYNAMICS REPRESENTATION

We instantiate the agent parameters θ_t as a frozen backbone Θ and a trainable reasoning adapter ϕ_t, yielding the task policy π_{Θ,ϕ_t}, where the backbone provides the latent-state interface and decoder, while the adapter controls the iterative reasoning transition. For a query x_i, let ϕ_i denote the reasoning adapter active when the query arrives, and let $Z_i^{(0)}$ be the initial latent state obtained from the backbone representation of x_i. The shallow task policy evolves the latent state for N_S steps:

$$Z_i^{(n+1)} = F_{\Theta,\phi_i}(Z_i^{(n)}; x_i), \quad n = 0, \ldots, N_S - 1, \tag{2}$$

where $F(\cdot)$ denotes the shared latent transition. The resulting sequence $\{Z_i^{(0)}, \ldots, Z_i^{(N_S)}\}$ records the internal reasoning trajectory that precedes the completion and is retained as part of the agent's experience ζ_t. To characterize the reasoning dynamics observed, the trajectory encoder processes the most recent recorded sequences in temporal order and summarizes how their latent states evolve across reasoning steps. These recorded trajectories characterize the latent dynamics induced by the deployed policy, but fail to reveal how the same queries would be processed under increased reasoning depth. At decision point t, we therefore sample a small diagnostic set $\mathcal{P}_t$ from the accumulated history ζ_t and reevaluate each query under the current adapter ϕ_t. For each $i \in \mathcal{P}_t$, the model executes both the standard N_S-step path and a deeper N_T-step path, where $N_T > N_S$. Let $Z_i^{S,n}(\phi_t)$ denote the shallow state at step n, with $Z_i^S(\phi_t) = Z_i^{S,N_S}(\phi_t)$ denoting its terminal state, and let $Z_i^T(\phi_t)$ denote the terminal state of the deep path. The operator Proj maps normalized latent states into a shared diagnostic space through a fixed row-orthogonal projection. We use the deep terminal state as the reference for measuring corrections along the shallow reasoning path:

$$\delta_{i,n} = \mathrm{sg}(\mathrm{Proj}(Z_i^T(\phi_t))) - \mathrm{Proj}(Z_i^{S,n}(\phi_t)), \quad n = 0, \ldots, N_S, \quad \delta_i = \delta_{i,N_S}, \tag{3}$$

where sg denotes stop-gradient. The profile $\{\delta_{i,0}, \ldots, \delta_{i,N_S}\}$ captures how the discrepancy between shallow and deep reasoning evolves along the shallow path, while the terminal correction δ_i represents the latent displacement between the shallow and deep terminal states. Since latent differences need not induce corresponding behavioral changes, we additionally compare the next-token distributions of the deep and shallow paths under the same token prefixes:

$$\kappa_i = \mathrm{Avg}_j \, D_{\mathrm{KL}}(\mathrm{sg}(p_{i,j}^{T,\tau}) \| p_{i,j}^{S,\tau}(\phi_t)), \tag{4}$$

where $p_{i,j}^{T,\tau}$ and $p_{i,j}^{S,\tau}(\phi_t)$ denote the temperature-scaled next-token distributions of the deep and shallow paths under the shared prefix, Avg_j averages over token positions, and τ is the temperature parameter. The resulting κ_i quantifies the output-level discrepancy induced by increased reasoning depth under the current adapter. A lightweight sequence encoder maps each latent trajectory to a query-level representation. For diagnostic queries, the representation additionally incorporates the correction profile and κ_i. A cross-query attention layer then aggregates the most recent M query representations together with their observed outcomes:

$$h_t = \mathrm{TrajEnc}_\psi(\zeta_t, \{\delta_{i,0:N_S}, \kappa_i\}_{i \in \mathcal{P}_t}). \tag{5}$$

where ψ denotes the learnable parameters of the trajectory encoder. The resulting representation h_t combines latent dynamics observed during normal operation with corrections revealed by deeper reasoning, providing trajectory-level evidence for evolution advantage prediction.

4.3 Evolution Advantage Prediction

The deep-to-shallow corrections characterize the changes induced by additional latent reasoning. We next examine whether these corrections can also be induced through changes to the reasoning adapter. For a candidate adapter ϕ, let $Z_i^S(\phi)$ denote the terminal state obtained by running the same N_S-step shallow path on query i. We define $\kappa_i(\phi)$ by replacing the shallow distribution in Eq. (4) with $p_{i,j}^{S,\tau}(\phi)$, such that $\kappa_i(\phi_t) = \kappa_i$. For each $i \in \mathcal{P}_t$, we define the probe objective as

$$\mathcal{L}_i^{\mathrm{p}}(\phi) = \tau^2 \kappa_i(\phi) + \left\| [\mathrm{Proj}(Z_i^S(\phi)) - \mathrm{Proj}(Z_i^S(\phi_t))] - \delta_i \right\|_2^2, \tag{6}$$

where the first term encourages the candidate shallow path to match the current deep path at the output level, while the bracketed difference captures the latent displacement induced by moving from ϕ_t to ϕ and matches it to the target correction δ_i. The deep-path distribution and correction target are fixed, so gradients are taken only through the shallow path.

We use the probe gradients across diagnostic queries to construct a low-dimensional local update subspace S_t that captures their dominant shared parameter directions. Let J_ϕ denote the Jacobian with respect to the adapter parameters. The first-order response of query i within this subspace is $G_i = J_\phi \big[\mathrm{Proj}(Z_i^S(\phi))\big]\big|_{\phi=\phi_t} S_t$. For a subspace coordinate u, $G_i u$ approximates the latent correction locally induced on query i, while δ_i is its target correction. We partition the diagnostic queries into a fitting subset $\mathcal{A}$ and a disjoint consistency subset $\mathcal{C}$ to assess whether a shared parameter change can account for the observed corrections beyond the queries used to estimate it. For any $\mathcal{S} \subseteq \mathcal{P}_t$, we define the correction fitting error and the shared local change as

$$E_{\mathcal{S}}(u) = \mathrm{Avg}_{i \in \mathcal{S}} \, \|G_i u - \delta_i\|_2^2, \; u_t^\star = \arg \min_{\|u\|_2 \le \epsilon} E_{\mathcal{A}}(u), \tag{7}$$

where ϵ limits the fit to the local regime, and $u_t^\star$ serves as a diagnostic of whether a common parameter change can explain the observed corrections. To determine whether updating is preferable to waiting, we construct two branches and fix their first actions to $a_t = 1$ and $a_t = 0$. Both branches then process the same query stream for H subsequent batches under a shared continuation policy. Let r_s^{ret} denote the mean verifier outcome on a fixed retention set at time s. For a prescribed first action a, we define the finite-horizon return and its associated evolution advantage as

$$\mathcal{J}_{t,H}(a) = \mathbb{E}\left[\frac{1}{H}\sum\nolimits_{k=1}^{H} r_{t+k} - (r_t^{\text{ret}} - r_{t+H}^{\text{ret}}) \mid \theta_t, \zeta_t, a_t = a\right], A_{t,H} = \mathcal{J}_{t,H}(1) - \mathcal{J}_{t,H}(0), \quad (8)$$

where $\mathcal{J}_{t,H}(a)$ measures future task performance while accounting for capability degradation over the same horizon. Since the wait branch may still trigger an update at later decision points, $A_{t,H} > 0$ indicates that updating at the current decision point is preferable to postponing the update. We use this paired advantage as the supervision target for a lightweight predictor based on the evidence available at the current decision point:

$$\widehat{A}_{t,H} = \text{MLP}_\omega(h_t, \underbrace{E_{\mathcal{A}}(0) - E_{\mathcal{A}}(u_t^\star)}_{\text{local realizability}}, \underbrace{E_{\mathcal{C}}(0) - E_{\mathcal{C}}(u_t^\star)}_{\text{cross-query consistency}}). \quad (9)$$

4.4 Impulse-Triggered On-Policy Self-Distillation

The evolution policy triggers OPSD when the predicted finite-horizon advantage favors an immediate update, $a_t = \mathbf{1}[\widehat{A}_{t,H} > 0]$. Otherwise, the adapter is retained until the next decision point. The zero boundary follows directly from the advantage definition in Eq. (8), without introducing an additional decision threshold. Once triggered, OPSD samples historical queries from ζ_t and has the shallow student regenerate completions at each optimization step, forming an on-policy batch $\mathcal{O}_\phi$. The teacher and student share the same query and initial latent state, but operate at different reasoning depths: $Z^T = F_{\Theta,\bar{\phi}}^{[N_T]}(Z^{(0)}; x)$, $Z^S(\phi) = F_{\Theta,\phi}^{[N_S]}(Z^{(0)}; x)$, where $N_T > N_S$, $[\cdot]$ denotes repeated latent transitions, and $\bar{\phi}$ is a frozen copy of the adapter ϕ_t. The teacher operates on student-generated prefixes and provides both next-token distributions and terminal latent targets. As the student changes during optimization, completions are resampled at each step to keep the supervision aligned with its current policy. Let $p_j^{T,\tau}$ and $p_j^{S,\tau}(\phi)$ denote the temperature-scaled teacher and student distributions at position j under the same student-generated prefix. Here, Avg^{tok} averages first over tokens within each completion and then over the batch. The behavioral distillation loss is

$$\mathcal{L}_{\text{b}}(\phi) = \text{Avg}_{\mathcal{O}_\phi}^{\text{tok}}\left[\tau^2 D_{\text{KL}}(\text{sg}(p_j^{T,\tau}) \| p_j^{S,\tau}(\phi))\right], \quad (10)$$

which transfers the deeper path's predictive behavior under the student's own prefixes. We further align the terminal latent states in the same diagnostic projection space:

$$\mathcal{L}_{\text{lat}}(\phi) = \text{Avg}_{\mathcal{O}_\phi}^{\text{qry}}\left[\| \text{Proj}(Z^S(\phi)) - \text{sg}(\text{Proj}(Z^T))\|_2^2\right], \quad (11)$$

where Avg^{qry} averages over sampled queries. This term encourages the shallow path to approximate the deep terminal representation within its limited reasoning depth. To retain previously acquired behavior, we maintain a replay memory $\mathcal{R}_t$ of verified agent-generated completions. On their recorded prefixes, the shallow student minimizes

$$\mathcal{L}_{\text{r}}(\phi) = -\,\text{Avg}_{\mathcal{R}_t}^{\text{tok}}[\log p_j^S(y_j^\star; \phi)], \quad (12)$$

where $p_j^S(y_j^\star; \phi)$ is the conditional probability assigned to replay token $y_j^\star$ under its corresponding prefix. The complete update objective is

$$\mathcal{L}_{\text{OPSD}}(\phi) = \mathcal{L}_{\text{b}}(\phi) + \alpha\mathcal{L}_{\text{lat}}(\phi) + \beta\mathcal{L}_{\text{r}}(\phi), \quad (13)$$

where α and β control latent distillation and capability retention, respectively.

5 Experiments

5.1 Experimental Setups

Datasets. We evaluate on seven benchmarks covering mathematical reasoning, code generation, general reasoning, and multi-hop question answering. **GSM8K** (Cobbe et al., 2021) contains grade-school math word problems, while **MATH** (Hendrycks et al., 2021) covers competition-level mathematics across diverse domains. For code generation, we use **HumanEval+** and **MBPP+** from

Table 1: Performance of LATCH and other baselines on various tasks and backbones. The best result in each column is marked in **bold**.

Methods	GSM8K	MATH	HumanEval+	MBPP+	BBH	HotpotQA		MuSiQue	
	EM ↑	EM ↑	Pass@1 ↑	Pass@1 ↑	Acc(%) ↑	EM ↑	F1 ↑	EM ↑	F1 ↑
LLaMA-3.1-8B									
VANILLA	78.1	41.4	59.8	61.4	58.2	34.5	43.7	23.4	34.2
PERIODIC	78.7	42.9	60.4	61.9	59.2	35.1	44.2	23.9	35.4
RANDOM	78.4	42.7	59.1	62.6	58.5	35.8	43.9	23.7	35.3
EARLY	79.2	43.4	61.0	62.4	59.1	35.7	44.8	24.4	36.0
ACE	80.3	44.5	60.4	62.7	60.2	39.2	48.3	28.0	39.3
REME	79.5	43.8	60.9	63.2	60.8	40.3	49.1	29.1	41.4
MATTS	83.6	49.2	65.2	67.4	63.6	39.1	48.4	28.4	40.9
EVOLVER	81.5	46.8	62.8	64.9	62.7	41.8	51.0	30.5	42.7
LATCH	**88.2**	**53.6**	**69.4**	**70.5**	**67.9**	**44.3**	**52.8**	**32.5**	**43.6**
Qwen3.8-27B									
VANILLA	92.3	75.2	79.3	76.2	77.6	50.4	59.3	40.6	50.6
PERIODIC	92.7	75.8	80.5	76.7	78.1	51.1	60.1	41.3	51.4
RANDOM	92.5	75.5	79.9	76.5	77.9	50.8	59.8	41.0	51.1
EARLY	93.0	76.5	81.1	77.2	78.7	51.8	60.8	42.0	52.2
ACE	93.2	77.2	80.5	78.0	79.8	56.0	64.5	46.2	56.8
REME	92.9	76.8	81.1	78.6	80.2	57.1	65.5	47.4	58.0
MATTS	94.5	81.5	85.4	83.1	84.0	56.3	65.0	46.7	57.2
EVOLVER	93.7	79.6	83.5	80.4	82.0	59.4	68.1	49.6	60.3
LATCH	**96.8**	**83.5**	**88.4**	**86.2**	**87.9**	**62.4**	**71.2**	**53.9**	**63.1**

EvalPlus (Liu et al., 2023), which extend HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) with substantially expanded unit tests. **BBH** (Suzgun et al., 2023) comprises challenging BIG-Bench reasoning tasks. **HotpotQA** (Yang et al., 2018) evaluates multi-hop QA over multiple documents under the distractor setting. **MuSiQue** (Trivedi et al., 2022) evaluates compositional multi-hop reasoning through connected single-hop steps.

Baselines. We compare LATCH with four update-timing baselines and four experience-learning methods. VANILLA processes the query stream without updates. PERIODIC updates at fixed intervals, RANDOM at randomly selected points, and EARLY near the beginning of the stream. ACE (Zhang et al., 2026c) incrementally refines a contextual playbook from agent experience. REME (Cao et al., 2026) builds procedural memory for future tasks. MATTS (Ouyang et al., 2026) combines memory-based learning with additional test-time exploration and consolidates resulting trajectories into reusable memory. EVOLVER (Wu et al., 2026b) distills interaction trajectories into strategic principles to evolve the agent policy.

Evaluation Metrics. We use benchmark-specific metrics for task performance. GSM8K and MATH are evaluated by exact match (EM) on normalized final answers, HumanEval+ and MBPP+ by Pass@1 under the EvalPlus test suites, and BBH by accuracy (Acc.). For HotpotQA and MuSiQue, we report both exact match and token-level F1. In the online task stream, each response is evaluated before its outcome is added to the agent's experience or used for subsequent updates.

Implementation Details. We use LLaMA-3.1-8B (Touvron et al., 2023) and Qwen3.8-27B (Qwen Team, 2026) as frozen backbones with rank-16 LoRA reasoning adapters. All experiments run in bfloat16 on an NVIDIA RTX PRO 6000 GPU. Each benchmark is organized into an online stream of experience batches. GSM8K examples are grouped by annotated arithmetic operations and reference-solution length, MATH by difficulty level, BBH by task category, HotpotQA by question type, and MuSiQue by reasoning hops; HumanEval+ and MBPP+ use seeded random permutations. Examples are shuffled within groups and batches are interleaved to induce recurring distribution shifts. Each query appears once, all methods share the same stream for a given seed, and an update–wait decision is made after each nonterminal batch. Stream-construction annotations are hidden from the evolution policy. The deployed policy uses $N_S = 2$ latent reasoning steps and the deeper path $N_T = 4$. At each decision point, the trajectory encoder processes the most recent $M = 64$ experiences; we sample $|\mathcal{P}_t| = 16$ diagnostic queries and retain $d = 4$ shared parameter directions for local realizability analysis. A sequence encoder with cross-query attention produces the trajectory representation, which a two-layer MLP combines with realizability and consistency signals to predict evolution advantage. The predictor is trained on paired update–wait rollouts from develop-

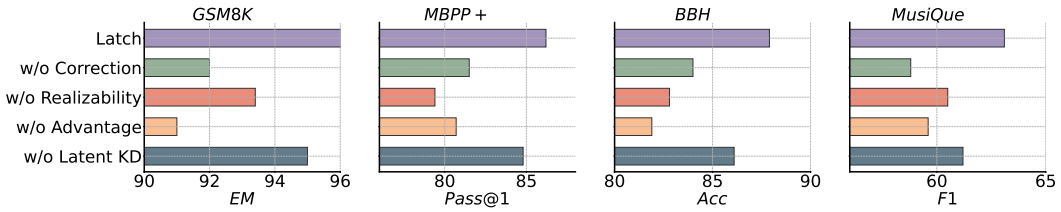
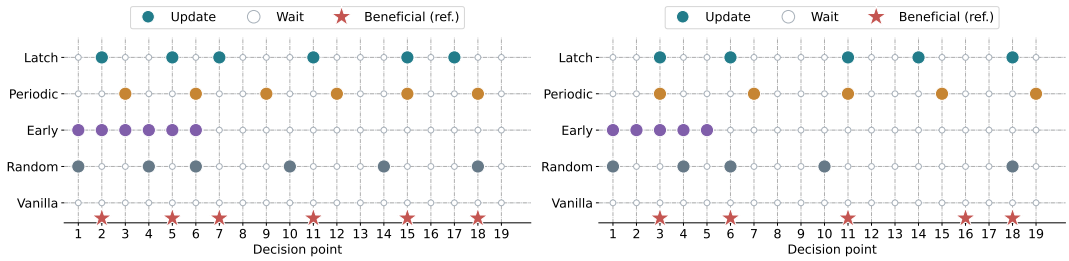


Figure 3: Ablation study across different representative reasoning tasks on Qwen3.8-27B.



(a) LLaMA-3.1-8B (b) Qwen3.8-27B

Figure 4: Evolution timing on GSM8K with different backbones.

ment streams disjoint from online evaluation, using identical future queries and random seeds for both branches. We set $H = 2$ experience batches and freeze the predictor during online evaluation. When updating, OPSD resamples student completions at each optimization step and performs eight AdamW steps with effective batch size eight and learning rate 10^{-4}. We set $\tau = 1.5$, $\alpha = 0.5$, and $\beta = 0.3$ for distillation temperature, latent distillation, and replay, respectively. All update-timing policies use the same initialized backbone and reasoning adapter.

5.2 Experimental Results

We compare our method with a range of baselines across two backbone models and diverse reasoning tasks, as shown in Table 1. LATCH consistently achieves the strongest overall performance across mathematical reasoning, code generation, general reasoning, and multi-hop question answering. The gains are preserved across both backbones, indicating that the proposed evolution policy is not tied to a particular model scale or architecture. The comparison with PERIODIC, RANDOM, and EARLY shows that simply changing when updates occur is insufficient: effective evolution requires identifying decision points at which accumulated reasoning experience is both informative and suitable for parameter internalization. LATCH also outperforms stronger adaptive and self-evolving baselines, suggesting that update timing benefits from explicitly modeling latent reasoning dynamics, deep-to-shallow corrections, and their parameter realizability rather than relying only on observable outcomes or generic update schedules. The improvements are consistent across heterogeneous task families. On mathematical and general reasoning tasks, LATCH benefits from recognizing recurring reasoning patterns that can be consolidated into the model. On code-generation and multi-hop QA benchmarks, where task-specific corrections and longer reasoning dependencies can make indiscriminate updates particularly harmful, selective evolution remains effective.

Ablation Study. In Table 3, we report ablation results across four representative tasks. Removing deep-to-shallow corrections or evolution advantage prediction generally leads to the largest performance degradation, indicating that effective evolution requires both process-level evidence of missing reasoning and an explicit comparison between updating and waiting. Omitting parameter-space realizability also consistently hurts performance, as the magnitude of a latent correction does not determine whether it can be induced by a shared parameter update. Removing latent distillation yields a smaller but consistent decline, showing that latent-state alignment transfers information beyond output-level distillation. The uniform advantage of the full model indicates that these components address distinct stages of the evolution process.

Evolution Decision Analysis. Figure 4 compares update decisions across all decision points on GSM8K. The red stars indicate points at which updating has a positive measured advantage on a

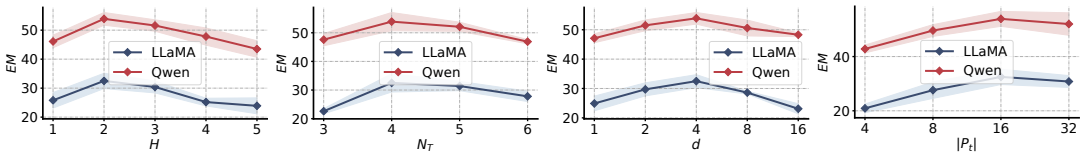


Figure 5: Hyperparameter analysis on MuSiQue under various backbones.

fixed reference trajectory; they do not represent optimal decisions shared by all methods. With LLaMA-3.1-8B, LATCH updates at five of the six marked points, while with Qwen3.8-27B it updates at four of the five marked points. Its decisions also change across backbones, consistent with an update policy that responds to the current model and its accumulated experience. PERIODIC and EARLY updating follow predetermined schedules and overlap with fewer marked points in both settings. This comparison provides evidence that LATCH identifies favorable opportunities on the reference trajectory.

Hyperparameter Sensitivity. We conduct hyperparameter analysis in Figure 5, evaluating the sensitivity of LATCH to four key hyperparameters on MuSiQue across two backbones. Increasing the rollout horizon H first improves performance by providing a more informative estimate of the update–wait consequence, but overly long horizons weaken credit assignment as the effect of the initial action becomes increasingly entangled with subsequent decisions. The teacher depth N_T exhibits a milder trade-off: deeper reasoning initially yields more informative deep-to-shallow corrections, whereas excessive depth enlarges the teacher–student gap and makes these corrections harder to internalize. The subspace dimension d is relatively stable around the default setting, with small values limiting the capacity to capture shared correction directions and large values introducing less consistent parameter directions. The diagnostic set size $|\mathcal{P}_t|$ is more sensitive when small, since insufficient queries lead to unreliable estimates of shared realizability and cross-query consistency; performance improves as more evidence is incorporated and then gradually saturates.

Efficiency and Cost Analysis. Table 2 reports the inference and evolution costs of different methods on MuSiQue. LATCH incurs moderate overhead over lightweight update strategies while remaining more efficient than methods that rely on frequent evolution or repeated LLM interactions. Selective triggering reduces latency and training cost by invoking OPSD only when the predicted evolution advantage is positive. This trend is consistent across both backbones, suggesting that the efficiency gain is not model-specific. Combined with the performance gains in Table 1, these results demonstrate a favorable performance–cost trade-off: LATCH concentrates additional computation on diagnostic reasoning and model updates when evolution is expected to be beneficial.

Table 2: Efficiency and computational cost on MuSiQue.

Methods	LLaMA-3.1-8B				Qwen3.8-27B			
	Avg. Latency ↓	*LLM Calls*↓	*Train Cost* ↓	*Total Cost* ↓	*Avg. Latency*↓	*LLM Calls*↓	*Train Cost* ↓	*Total Cost* ↓
VANILLA	3.42	1.00	–	2.30	9.68	1.00	–	6.50
PERIODIC	4.84	1.24	0.88	3.25	13.49	1.24	2.31	9.06
RANDOM	4.93	1.24	0.91	3.31	13.72	1.24	2.39	9.21
EARLY	4.59	1.24	0.76	3.08	12.96	1.24	2.02	8.70
ACE	8.91	3.00	3.34	5.98	25.83	3.00	9.61	17.34
REME	4.01	1.34	0.39	2.69	11.20	1.34	1.02	7.52
MATTS	22.84	6.00	1.07	15.33	65.37	6.00	3.08	43.89
EVOLVER	31.86	9.00	15.37	21.39	89.46	9.00	45.62	60.06
LATCH	**4.68**	**1.78**	**0.84**	**3.14**	**13.11**	**1.78**	**2.30**	**8.80**

6 CONCLUSION

In this work, we investigate when model evolution is worthwhile for long-horizon LLM agents and introduce LATCH, a selective evolution framework based on latent reasoning dynamics. LATCH extracts deep-to-shallow reasoning corrections, assesses whether they can be jointly realized through shared parameter changes, and uses these signals to predict the finite-horizon advantage of updating now over waiting. When evolution is triggered, on-policy self-distillation transfers deeper reasoning into the shallow policy while replay preserves prior capabilities. Experiments across diverse reasoning benchmarks and model backbones show that LATCH improves cumulative performance and capability retention while reducing unnecessary updates.

AI USE STATEMENT

This work is entirely original and was conducted by the authors. Generative AI tools were not used to produce any content of the work; they were used solely to assist with language refinement and to improve the clarity and quality of the text.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Huan ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru WANG, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, 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. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=CTr3bovS5F. Survey Certification.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16803–16822, 2026.

Arthur Chen, Zuxin Liu, Jianguo Zhang, Akshara Prabhakar, Zhiwei Liu, Shelby Heinecke, Silvio Savarese, Victor Zhong, and Caiming Xiong. Test-time adaptation for llm agents via environment interaction. In *International Conference on Learning Representations*, volume 2026, pp. 39256–39288, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Sijia Chen and Baochun Li. Toward adaptive reasoning in large language models with thought rollback. *arXiv preprint arXiv:2412.19707*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Zijie Dai, Shiyuan Deng, Sheng Guan, Yizhou Tian, Xin Yao, Xiao Yan, and James Cheng. Recmem: Recurrence-based memory consolidation for efficient and effective long-running llm agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 32353–32376, 2026.

Aniket Didolkar, Nicolas Ballas, Sanjeev Arora, and Anirudh Goyal. Metacognitive reuse: Turning recurring llm reasoning into concise behaviors. *arXiv preprint arXiv:2509.13237*, 2025.

Zhuoyun Du, Runze Wang, Huiyu Bai, Zouying Cao, Xiaoyong Zhu, Yu Cheng, Bo Zheng, Wei Chen, and Haochao Ying. Enabling agents to communicate entirely in latent space. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27106–27129, 2026.

Yujie Feng, Hao Wang, Jian Li, Xu Chu, Zhaolu Kang, Yiran Liu, Yasha Wang, Philip S Yu, and Xiao-Ming Wu. Forever: Forgetting curve-inspired memory replay for language model continual learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24945–24965, 2026.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2025.

Bincheng Gu, Min Gao, Zongwei Wang, Yibing Bai, Yulan He, and Junliang Yu. Beyond confidence: Stability-aware test-time adaptation for llm reasoning. *arXiv preprint arXiv:2609.11393*, 2026.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Itxz7S4Ip3.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-STar: Training verifiers for self-taught reasoners. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=stmqBSW2dV.

Minki Kang, Jongwon Jeong, Seanie Lee, Jaewoong Cho, and Sung Ju Hwang. Distilling llm agent into small models with retrieval and code tools. *Advances in Neural Information Processing Systems*, 38:106501–106538, 2025.

Yeskendir Koishekenov, Aldo Lipani, and Nicola Cancedda. Encode, think, decode: Scaling test-time reasoning with recursive latent thoughts. *arXiv preprint arXiv:2510.07358*, 2025.

Kyumin Lee, Minjin Jeon, Sanghwan Jang, and Hwanjo Yu. Steper: Step-wise knowledge distillation for enhancing reasoning ability in multi-step retrieval-augmented language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 29489–29511, 2025.

Rui Li and Shuang Cao. From trajectories to graphs: Contract-checked editing for verifier-guided llm reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 43259–43306, 2026.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Xiangyu Peng, Congying Xia, Xinyi Yang, Caiming Xiong, Chien-Sheng Wu, and Chen Xing. Regenesis: Llms can grow into reasoning generalists via self-improvement. In *International Conference on Learning Representations*, volume 2025, pp. 62484–62519, 2025.

Adam Piaseczny, Md Kamran Chowdhury Shisher, Shiqiang Wang, and Christopher Brinton. Rccda: Adaptive model updates in the presence of concept drift under a constrained resource budget. *Advances in Neural Information Processing Systems*, 38:154900–154944, 2025.

Fuli Qiao and Mehrdad Mahdavi. Merge before forget: A single lora continual learning via continual merging. In *International Conference on Learning Representations*, volume 2026, pp. 8133–8159, 2026.

Xihe Qiu, Leijun Cheng, Teqi Hao, and Xiaoyu Tan. Arma: Mitigating catastrophic forgetting using attention-regularized model averaging in continual fine-tuning large language models. *IEEE Transactions on Artificial Intelligence*, 2025.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

Shuai Shao, Qihan Ren, Dongrui Liu, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, et al. Your agent may misevolve: Emergent risks in self-evolving llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 99728–99793, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

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

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Rodriguez Aurelien, Joulin Armand, Grave Edouard, and Lample Guillaume. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL https://formacion.actuarios.org/wp-content/uploads/2024/05/2302.13971-LLama-Open-and-Efficient-Foundation-Language-Models.pdf.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 05 2022. ISSN 2307-387X. doi: 10.1162/tacl_a_00475. URL https://doi.org/10.1162/tacl_a_00475.

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

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

Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. Group-evolving agents: Open-ended self-improvement via experience sharing. *arXiv preprint arXiv:2602.04837*, 2026.

Bin Wu, Edgar Meij, and Emine Yilmaz. Beyond static toolsets: Self-evolving llm tool agents via continual documentation adaptation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 21519–21539, 2026a.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. From interactions to principles: Experience-driven self-distillation for evolving LLM agents. In *Forty-third International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=BamDfCP5R3.

ShiLin Xiao, Tianxiang Xu, Canran Xiao, Weihao Luo, Liwei Hou, and Chuangxin Zhao. Meta-ucf: Unified task-conditioned lora generation for continual learning in large language models. In *The Fourteenth International Conference on Learning Representations*, 2026.

Zehao Xiao and Cees GM Snoek. Beyond model adaptation at test time: A survey. *arXiv preprint arXiv:2411.03687*, 2024.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Watch every step! llm agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1556–1572, 2024.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts llm agents: An empirical study of experience-following behavior. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 623–645, 2026.

Cheng Yang, Xuemeng Yang, Licheng Wen, Daocheng Fu, Jianbiao Mei, Rong Wu, Pinlong Cai, Yufan Shen, Nianchen Deng, Jia Xu, et al. Towards self-evolving agents: Enabling autonomy through interactive experience refinement. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 30424–30451, 2026a.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Minkai Xu, Joseph E Gonzalez, Bin Cui, and Shuicheng Yan. Supercorrect: Advancing small llm reasoning with thought template distillation and self-correction. In *International Conference on Learning Representations*, volume 2025, pp. 3386–3418, 2025.

Shuo Yang, Caren Han, Xueqi Ma, Yan Li, Mohammad Reza Ghasemi Madani, and Eduard Hovy. Evotool: Self-evolving tool-use policy optimization in llm agents via blame-aware mutation and diversity-aware selection. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 43553–43572, 2026b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018. URL https://aclanthology.org/D18-1259.

Ye Yu, Xiaopeng Yuan, Haibo Jin, Heming Liu, Yaoning Yu, and Haohan Wang. Do self-evolving agents forget? capability degradation and preservation in lifelong llm agent adaptation. *arXiv preprint arXiv:2605.09315*, 2026a.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21457–21483, 2026b.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification. In *Third Conference on Language Modeling*, 2026a. URL https://openreview.net/forum?id=gQjmJIichQ.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026b.

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

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Junhao Zheng, Chengming Shi, Xidi Cai, Qiuke Li, Duzhen Zhang, Chenxing Li, Dong Yu, and Qianli Ma. Lifelong learning of large language model based agents: A roadmap. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart J Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. *Advances in Neural Information Processing Systems*, 38:79931–79963, 2025.

A Latent Reasoning Path

For an input query, we process its token embeddings with the frozen backbone and use the contextual hidden representation at the final query position to obtain the initial latent state $Z^{(0)}$. The shallow and deep paths start from this same state and apply the transition in Eq. (2) for N_S and N_T steps, respectively. At each step, the current latent state is appended to the query and preceding latent states as a continuous embedding. We run the backbone on this extended sequence, extract the hidden representation at the newly appended position, and apply the reasoning adapter to obtain the next latent state. Specifically, the online adapter contributes a low-rank residual to this representation at each latent transition; it acts on the recurrent latent state rather than on the backbone's attention or feed-forward projections. After the final step, we append the terminal latent state to the query as a continuous embedding and generate the completion autoregressively with the backbone decoder. For diagnostic comparisons, Proj is applied to the resulting latent states, including the shallow and deep terminal states. We first apply layer normalization along the hidden dimension, flatten the state, and scale it by the square root of its dimensionality. We then map it into the shared diagnostic space using a projection constructed by drawing a Gaussian matrix and orthogonalizing its columns with QR decomposition. This projection is generated once with a fixed random seed and reused across queries, reasoning depths, and update steps.

B Decision-Point Evidence Construction

At a decision point, we retrieve the most recent M experiences and use the shallow latent trajectories recorded when those queries were originally answered. Each trajectory therefore reflects the adapter that was active at the time of the query. To obtain diagnostic evidence under the current adapter ϕ_t, we select the most recent $|\mathcal{P}_t|$ queries from the available experience and rerun each query along both the N_S-step shallow path and the N_T-step deep path. These reruns produce the correction profile $\{\delta_{i,n}\}_{n=0}^{N_S}$ defined in § 4.2. For the output-level comparison, we generate a completion with the current shallow policy and score its token prefixes with both paths. The prefixes are thus identical in the two evaluations, and their next-token distributions yield κ_i in Eq. (4).

To construct h_t, the trajectory encoder first summarizes each recorded shallow path from its latent states and their changes across reasoning steps. For a diagnostic query, a separate sequence encoder summarizes its correction profile, and the resulting representation also includes κ_i. We associate each historical query with its recorded verifier outcome, its position in the recent experience window, and whether it was originally processed by an earlier adapter. The available trajectory, diagnostic, and historical features are combined into a query-level representation; queries outside $\mathcal{P}_t$ contribute their recorded trajectories and outcomes without diagnostic features. An attention layer assigns weights to the M query representations and aggregates them into the decision-point representation h_t used for evolution advantage prediction.

C Paired Evolution Advantage Estimation

We construct predictor training examples at decision points in development streams disjoint from the online evaluation streams. At each point, we compute the trajectory and realizability features from the experience available before the decision, then copy the same agent state, including the current adapter, experience history, and replay memory, into two rollout branches. The first branch is forced to update immediately with OPSD, while the second is forced to wait. Both branches receive the same subsequent experience batches over horizon H. After the first action, they follow the same continuation policy, which evaluates each branch's current state at later decision points. We use a fixed continuation policy in the initial round of label generation. In subsequent rounds, the predictor trained in the preceding round is frozen and used as the continuation policy for both branches; the resulting labels are used to train the next predictor.

We pair stochastic operations across the two branches by using the same random seed for each rollout replicate and the same seed at corresponding update opportunities. For each branch, we average the verifier outcomes within each future batch and compute the finite-horizon return in Eq. (8). To measure retention, we select a fixed set of held-out, verified queries from the pre-online data, disjoint from the continuation batches. We evaluate this same set before branching and at the end of each

Algorithm 1 LATCH: Selective Agent Evolution

Require: Frozen backbone Θ, initialized adapter ϕ_0, development streams, online batches $\{\mathcal{B}_t\}_{t=1}^{T}$

Initialize a fixed continuation policy
for each predictor-training round **do**
  for each development decision point reached under the continuation policy with H subsequent batches available **do**
    Construct trajectory and realizability evidence from the current experience
    Fork identical states; force `update` and `wait` as their first actions
    Continue both branches on the same next H batches under the shared continuation policy, with paired random seeds
    Compute the paired target $A_{t,H}$ using Eq. (8)
  end for
  Train $\mathrm{TrajEnc}_{\psi}$ and MLP_{ω} on the paired targets
  Use the frozen predictor's update–wait rule as the next continuation policy
end for
Freeze the final predictor; set $\phi \leftarrow \phi_0$ and initialize online experience and replay memory
for $t = 1, \ldots, T$ **do**
  Answer and verify all queries in $\mathcal{B}_t$; record their experiences and maintain replay memory
  if $t < T$ **then**
    Reevaluate diagnostic queries under ϕ to obtain $\delta_{i,0:N_S}$ and κ_i
    Compute h_t, local realizability, and cross-query consistency
    Predict $\widehat{A}_{t,H}$ using Eq. (9)
    if $\widehat{A}_{t,H} > 0$ **then**
      Update ϕ with OPSD using Eq. (13)
    end if
  end if
end for

rollout; its mean verifier outcome supplies the retention term in the return. The update-branch return minus the wait-branch return gives one paired estimate of $A_{t,H}$. When multiple replicates are used, their mean provides the regression target, and their variance records the uncertainty of that estimate.

The predictor receives only features computed before the branches diverge: h_t, local realizability, and cross-query consistency. We standardize the advantage targets using statistics from the predictor training split and fit the predictor with a Huber regression loss. When labels have multiple rollout replicates, estimates with greater replicate variance receive lower weight. Model selection uses held-out development streams, and the resulting predictor is frozen before online evaluation. During online evaluation, its predicted advantage $\widehat{A}_{t,H}$ determines the update–wait action.

D Detailed Experimental Settings

D.1 Dataset Descriptions

Our evaluation uses seven datasets spanning four task types: mathematical reasoning, code generation, general reasoning, and multi-hop question answering. The mathematical datasets range from grade-school word problems to competition-level questions across several subject areas. The code-generation datasets specify Python programming tasks and provide tests that exercise the required behavior. The remaining datasets include heterogeneous reasoning problems and questions whose answers depend on evidence distributed across multiple documents. We describe the content and distinguishing characteristics of each dataset below.

GSM8K. GSM8K (Cobbe et al., 2021) consists of grade-school mathematics word problems written by human annotators. It contains 7,473 training problems and 1,319 test problems. Each problem describes a situation in natural language, often involving several quantities and relationships that must be identified before a calculation can be performed. Although the required operations are elementary—primarily addition, subtraction, multiplication, and division—solving a problem typically involves multiple dependent steps. For example, an intermediate quantity may need to be

calculated before it can be used to determine the final answer. Each example provides a worked solution that records the calculations and ends with a numerical answer. The dataset therefore combines comprehension of a word problem with the ability to maintain a consistent sequence of arithmetic operations.

MATH. MATH (Hendrycks et al., 2021) contains 12,500 problems from high-school mathematics competitions, divided into 7,500 training and 5,000 test problems. It covers seven subject areas: prealgebra, algebra, intermediate algebra, geometry, number theory, counting and probability, and precalculus. Problems are also assigned difficulty levels from one to five. Each example includes a complete step-by-step solution and a final answer, commonly expressed as a mathematical expression rather than a single integer. Depending on the subject, a solution may involve manipulating equations, applying geometric properties, constructing a counting argument, or combining several mathematical results. The range of subjects and difficulty levels distinguishes MATH from arithmetic-focused word-problem datasets: the solver must identify an appropriate approach as well as carry out the subsequent derivation.

HumanEval+. HumanEval+ is the EvalPlus (Liu et al., 2023) extension of HumanEval (Chen et al., 2021). HumanEval contains 164 Python function-completion problems. A problem supplies the beginning of a function, including its signature and a docstring describing the desired behavior, and asks for an implementation of that function. The specifications cover varied operations on numbers, strings, lists, and other Python objects; a correct implementation must satisfy the behavior described by the docstring, including cases beyond any examples it contains. HumanEval includes reference implementations and unit tests for its problems. HumanEval+ keeps the underlying programming tasks but substantially expands the test suites, introducing many more inputs for each function. This expansion helps distinguish implementations that satisfy the general specification from those that happen to work on only a small set of familiar cases.

MBPP+. MBPP+ extends Mostly Basic Programming Problems (MBPP) (Austin et al., 2021) through EvalPlus (Liu et al., 2023). The original MBPP dataset contains 974 short Python programming tasks intended to be accessible to entry-level programmers. Each task has a natural-language description and example test cases that illustrate the requested behavior. The problems involve basic program synthesis across a range of common operations, including numerical calculations, string processing, and manipulation of Python data structures. The examples clarify the intended input–output relationship but need not cover every case that a correct program should handle. MBPP+ builds on a manually curated subset of these tasks and adds a substantially larger collection of unit tests. As with HumanEval+, the expanded tests expose programs that reproduce the provided examples without correctly implementing the more general task description.

BBH. BIG-Bench Hard (BBH) (Suzgun et al., 2023) is a collection of 23 tasks selected from BIG-Bench. At the time of its construction, these tasks were among those on which evaluated language models had not surpassed average human-rater performance. BBH does not follow a single question template: its tasks include logical deduction, tracking shuffled objects, reasoning about dates, interpreting sequences of operations, and resolving ambiguities in language. The information needed to answer a question is generally contained in its prompt, but reaching the answer may require maintaining several intermediate states or applying multiple constraints in the correct order. Task formats also vary, with some questions requesting a choice from specified options and others requiring a short generated answer. This heterogeneity makes BBH a collection of distinct reasoning challenges rather than a benchmark centered on one domain or operation.

HotpotQA. HotpotQA (Yang et al., 2018) contains approximately 113,000 question–answer pairs constructed from Wikipedia articles. Its questions are designed to require information from multiple documents rather than a single passage. In a bridge question, information in one document identifies an intermediate entity that must be investigated in another document. In a comparison question, the answer depends on retrieving facts about two entities and determining how those facts relate. Each example provides an answer together with sentence-level supporting-fact annotations that identify the evidence required for the reasoning chain. HotpotQA offers a distractor setting, in which the relevant documents are mixed with unrelated documents, and a full-wiki setting, in which evidence must be located within the broader Wikipedia collection. Thus, the questions involve both identifying relevant evidence and combining it across documents.

MuSiQue. MuSiQue (Trivedi et al., 2022) is a multi-hop question-answering dataset constructed by composing single-hop questions into connected reasoning chains. During construction, an answer obtained at one step becomes necessary for resolving another step; the final question is therefore intended to depend on the intervening information rather than on one isolated passage. MuSiQue-Ans contains approximately 25,000 answerable questions with two to four reasoning hops and several composition structures. Questions are accompanied by passages that contain the necessary evidence as well as distractor passages, requiring the solver to identify and connect the relevant information. The dataset also provides MuSiQue-Full, which adds unanswerable contrast questions created by withholding information needed at an intermediate step. These design choices target disconnected reasoning, in which a system reaches an answer without following the full chain of evidence.

Together, the seven datasets cover different sources of difficulty: translating natural-language descriptions into calculations or programs, selecting and combining mathematical techniques, maintaining intermediate states across varied reasoning tasks, and integrating evidence across documents. Their distinct problem formats and annotation structures provide a broad characterization of the tasks considered in this work.

D.2 Comparison Baselines

We consider four controls for update timing and four methods for learning from agent experience. The timing controls vary when an adapter update is performed, while the experience-learning methods differ in how they extract, retain, and reuse information from past interactions. We describe each baseline below.

- Vanilla uses the initialized agent throughout the query stream without updating its reasoning adapter or adding an experience-learning mechanism. The agent processes each incoming query with the same policy, so completed tasks do not change its behavior on later queries. This baseline establishes the performance of the underlying backbone and its initial adapter in the absence of online evolution. It also provides a reference for the computational cost incurred by methods that adapt during the stream.
- Periodic performs an adapter update at fixed intervals. Its decisions depend only on the position in the stream: once the interval is specified, the update points are determined in advance and do not depend on task outcomes or the content of recent experiences. The policy therefore distributes updates across the stream but cannot respond to changes in the potential value of adaptation. Comparing against Periodic tests whether selecting update times from accumulated experience is more useful than following a predetermined schedule.
- Random selects update points at random, without using feedback or features of the accumulated experience to guide the selection. Unlike Periodic, its updates need not be evenly spaced, allowing the comparison to separate the effect of a regular schedule from that of making updates at arbitrary points. The resulting decisions may occur during both useful and unproductive periods of adaptation, providing a control for improvements attributable merely to performing updates.
- Early concentrates adapter updates near the beginning of the stream and subsequently retains the resulting adapter. This schedule tests the assumption that adaptation is most valuable when the agent has encountered relatively few tasks and can use the learned changes throughout the remainder of the stream. Because its decisions are fixed in advance, it does not account for useful patterns or changes in task demands that emerge later.
- ACE (Zhang et al., 2026c) treats an agent's context as an evolving playbook of task-relevant strategies. Its workflow separates experience generation, reflection on successful and unsuccessful behavior, and curation of the resulting insights. Rather than repeatedly rewriting the entire playbook, ACE applies structured, incremental edits that retain useful details while controlling redundancy. The updated playbook is supplied as context for subsequent tasks. ACE thus provides a comparison with experience-driven adaptation through the agent's external context rather than its model parameters.
- ReMe (Cao et al., 2026) maintains a procedural memory containing reusable "how-to" knowledge derived from agent interactions. It distills experience through success-pattern extraction, failure analysis, and comparisons between outcomes. For a new task, it retrieves

relevant memories using context-sensitive indexing; as more experience becomes available, utility-based refinement adds useful entries and removes outdated ones. This memory lifecycle goes beyond storing raw trajectories or accumulating an append-only collection of notes. ReMe provides a comparison with an agent that improves future decisions by refining and retrieving procedural knowledge.

- MATTS (Ouyang et al., 2026) extends ReasoningBank, which distills reusable reasoning strategies from successful and failed agent trajectories. ReasoningBank retrieves relevant strategies to guide a new task and incorporates lessons from the resulting interaction into its memory. MaTTS allocates additional test-time computation to generate multiple trajectories or refinement steps for a task. It uses the resulting contrast between successful and unsuccessful explorations to construct more informative memory items, while the existing memory guides further exploration. This baseline combines experience-based memory evolution with increased test-time exploration rather than selecting when to update a reasoning adapter.
- EVOLVER (Wu et al., 2026b) organizes agent improvement as a closed-loop experience lifecycle. An offline self-distillation stage converts interaction trajectories into a repository of abstract, reusable strategic principles. During online interaction, the agent retrieves relevant principles to guide its behavior and accumulates new trajectories. A policy reinforcement mechanism then uses interaction outcomes to iteratively improve the agent. EvolveR therefore provides a comparison with a method that couples explicit strategy distillation and retrieval with policy evolution, rather than deciding whether a particular batch of experience warrants an immediate adapter update.

D.3 METRIC DESCRIPTIONS

We report task performance using the metrics associated with each benchmark. All scores in the main results table are expressed as percentages, with higher values indicating better performance. Metrics are computed from the agent's response to each query and then aggregated over the corresponding benchmark.

Exact Match for Mathematical Reasoning. For GSM8K and MATH, exact match (EM) measures whether the final answer extracted from a generated response matches the reference answer after benchmark-specific normalization. The intermediate explanation is not scored independently: a response receives credit only when its final answer is correct. For GSM8K, this comparison concerns the numerical answer to a word problem. MATH answers can also contain symbolic expressions, so answer extraction and normalization must preserve the mathematical content of the final expression. For a benchmark with N problems, EM is the percentage of problems whose normalized predicted and reference answers match:

$$\mathrm{EM} = \frac{100}{N} \sum_{i=1}^{N} \mathbb{I}[\mathrm{norm}(\hat{a}_i) = \mathrm{norm}(a_i)] \,. \tag{14}$$

where $\hat{a}_i$ and a_i denote the extracted prediction and reference answer, respectively.

Pass@1 for Code Generation. For HumanEval+ and MBPP+, we report Pass@1 using the expanded EvalPlus test suites. A generated program is counted as correct only if it executes successfully and passes all tests associated with its task, including the additional tests introduced by EvalPlus. A solution that passes the original tests but fails an added test is therefore counted as incorrect. With one generated candidate per task, Pass@1 is the percentage of tasks for which that candidate passes the complete test suite:

$$\mathrm{Pass@1} = \frac{100}{N} \sum_{i=1}^{N} \mathbb{I}[\hat{c}_i \text{ passes all tests for task } i] \,. \tag{15}$$

where $\hat{c}_i$ is the generated program for task i. This execution-based measure evaluates program behavior rather than textual similarity to a reference implementation.

Accuracy for BBH. For BBH, accuracy (Acc.) records whether the answer to each reasoning problem is correct. The generated response is reduced to the answer required by the corresponding task and compared with its reference answer; an incorrect answer receives no partial credit even if the

response contains intermediate reasoning. Because BBH contains tasks with different output formats, answer extraction follows the format of each task. The reported accuracy aggregates these per-example correctness judgments over the evaluated BBH problems.

Answer Exact Match for Multi-hop Question Answering. For HotpotQA and MuSiQue, answer EM measures whether the predicted answer matches a reference answer after text normalization. Unlike mathematical EM, this normalization operates on answer text, typically lowercasing it and removing punctuation, articles, and excess whitespace. The entire normalized answer must match for the example to receive credit. For MuSiQue examples with multiple accepted answer aliases, the prediction is compared against the available references and receives credit if it matches any of them. We report answer EM rather than exact match on supporting-fact annotations or reasoning traces.

Token-level F1 for Multi-hop Question Answering. Token-level F1 measures partial overlap between a predicted answer and a reference answer. After text normalization, precision is the fraction of predicted tokens that occur in the reference, and recall is the fraction of reference tokens recovered by the prediction. Their harmonic mean gives the example-level score:

$$P_i = \frac{|T(\hat{a}_i) \cap T(a_i)|}{|T(\hat{a}_i)|}, R_i = \frac{|T(\hat{a}_i) \cap T(a_i)|}{|T(a_i)|}, F_{1,i} = \frac{2P_iR_i}{P_i + R_i}. \tag{16}$$

where token overlap counts repeated tokens according to their frequencies. F1 is zero when the answers share no tokens; cases involving empty answers follow the benchmark scorer. When multiple reference answers or aliases are available, the highest score is used. The reported F1 is the mean of the example-level scores, multiplied by 100. Together, answer EM and F1 distinguish fully correct answers from answers that recover only part of the reference text.

Scoring in the Online Stream. Each query is scored using the response produced by the agent's policy at the time that query is encountered. The response and its outcome are recorded before the outcome is added to the experience history or used in a subsequent update decision. An update made after a batch can affect future responses, but it does not change the scores already assigned to completed queries. The reported task metrics therefore aggregate responses produced during the online stream rather than reevaluating earlier queries with the final adapted policy.

D.4 Stream Construction

We construct a separate online experience stream for each benchmark. Stream construction consists of three stages: organizing examples by dataset-specific attributes, forming batches within each group, and interleaving the resulting batches.

For GSM8K, we include all 1,319 questions in the test split. We characterize each question using the number of annotated arithmetic operations and the length of its reference solution. Because these quantities have different scales, we rank questions separately by each quantity and combine the two normalized ranks into a difficulty score. Questions with similar scores are placed in the same difficulty group. The operation count captures the amount of explicit calculation, while solution length provides a complementary indication of the reasoning needed to reach the answer. For MATH, we use the dataset's provided difficulty levels directly as groups, without deriving new labels from model predictions. For BBH, each task category forms a group, preserving the distinction among its different types of reasoning problems. HotpotQA questions are grouped by their annotated type, bridge or comparison. MuSiQue questions are grouped by the number of reasoning hops required to reach the answer. These attributes describe the structure of the questions and are used only to organize the stream. HumanEval+ and MBPP+ are handled differently: because we do not assign their programming problems difficulty groups, we generate a seeded random permutation for each dataset and form batches directly from that permutation.

Within each group, we shuffle the examples using a fixed random seed and divide the shuffled sequence into consecutive experience batches. A batch therefore contains examples from one group. For benchmarks with multiple groups, we cycle through groups with remaining batches and take one batch from each group per cycle. Once a group is exhausted, subsequent cycles continue over the remaining groups. This interleaving prevents the stream from presenting all examples of one type in a single uninterrupted segment and allows previously encountered types to recur later. The code-generation streams require no interleaving because they are formed from a single random permutation.

The resulting batch contents, batch boundaries, and batch order are identical for all methods evaluated on a benchmark with the same seed. Each query appears exactly once. The agent processes every query in a batch with the policy available at the start of that batch and does not update during batch execution. After the batch is complete, its responses are scored and the corresponding trajectories and feedback become available as experience. Every boundary after a nonterminal batch is then an update–wait decision point for the update-timing methods. An update can affect responses to later batches, whereas a wait decision retains the current adapter. For a stream containing T batches, this yields $T - 1$ decision opportunities.

Dataset annotations used for grouping, including GSM8K reference-solution properties, are not supplied as inputs to the evolution policy. At each decision point, the policy has access only to the completed experience and feedback available up to that boundary; future queries and their outcomes remain unavailable. Reference solutions used to organize the stream are not used as answers or adaptation targets before their associated queries have been processed.

D.5 Details of the Ablation Study

We evaluate four ablations of LATCH to examine the contributions of deep-to-shallow correction evidence, shared parameter realizability, evolution-advantage prediction, and latent-state distillation. The full model includes all four components. All variants use the same backbone, initial adapter, benchmark stream, diagnostic-query budget, and OPSD optimization settings unless the ablated component requires a change. Predictors whose inputs or update operators differ from those of the full model are trained on the corresponding development streams.

- *Without Correction.* We remove the correction profile $\{\delta_{i,n}\}_{n=0}^{N_S}$ and output-distribution discrepancy κ_i from the query representations used to construct h_t. The trajectory encoder therefore represents recorded shallow reasoning trajectories and verifier outcomes without these direct correction features. The deep and shallow diagnostic paths remain available for computing the realizability and consistency scores, and the deep teacher remains available for OPSD. We retrain the advantage predictor using the reduced trajectory representation.
- *Without Realizability.* We remove the local realizability score $E_{\mathcal{A}}(0) - E_{\mathcal{A}}(u_t^\star)$ and cross-query consistency score $E_{\mathcal{C}}(0) - E_{\mathcal{C}}(u_t^\star)$ from the advantage predictor. Its input consists only of the trajectory representation h_t, which still includes the deep-to-shallow correction evidence. The local subspace fitting procedure is omitted at decision time. We retrain the predictor on paired update–wait rollouts using this reduced input.
- *Without Advantage.* We replace the learned finite-horizon advantage predictor with a decision rule based on the realizability diagnostics. At each decision point, we compute
$$s_t = \big[E_{\mathcal{A}}(0) - E_{\mathcal{A}}(u_t^\star)\big] + \big[E_{\mathcal{C}}(0) - E_{\mathcal{C}}(u_t^\star)\big]. \tag{17}$$
OPSD is triggered when s_t exceeds a threshold calibrated on development streams. This variant retains the diagnostic paths, local subspace analysis, and OPSD update operator, but does not use paired finite-horizon rollouts to train an update–wait predictor.
- *Without Latent KD.* We remove the terminal latent-state alignment term from OPSD by setting $\alpha = 0$. The update objective becomes
$$\mathcal{L}_{\mathrm{OPSD}}^{-\mathrm{lat}}(\phi) = \mathcal{L}_{\mathrm{b}}(\phi) + \beta\mathcal{L}_{\mathrm{r}}(\phi). \tag{18}$$
Behavioral distillation from the frozen deep teacher, on-policy resampling of student completions, and replay retention remain unchanged. We generate paired development rollouts using this ablated update operator and train its advantage predictor on the resulting update–wait targets.

E Additional Experimental Results

E.1 Ablation Studies

Figure 6 reports additional ablation results on GSM8K, MBPP+, BBH, and MuSiQue with LLaMA-3.1-8B. The full LATCH model consistently outperforms all ablated variants, while the relative impact of each component varies across tasks. Removing deep-to-shallow correction produces the

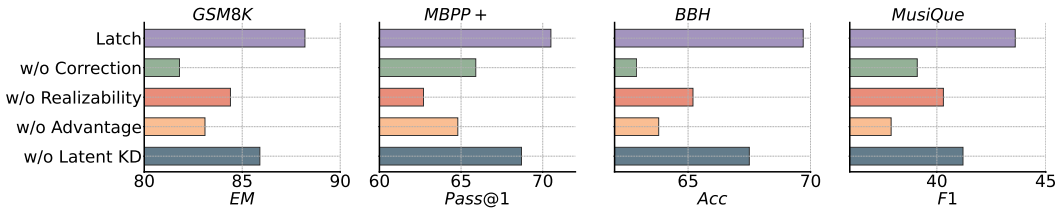
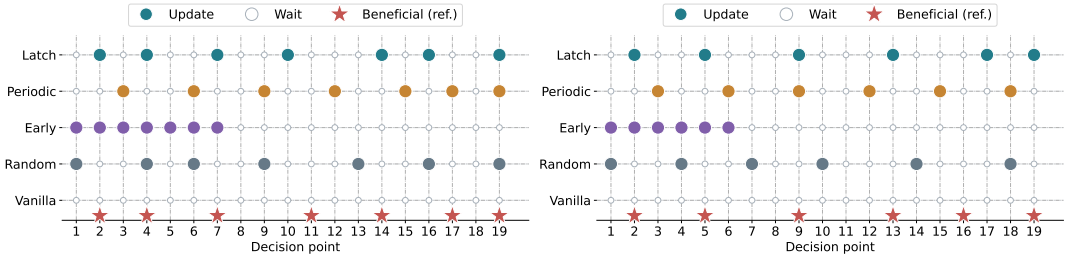


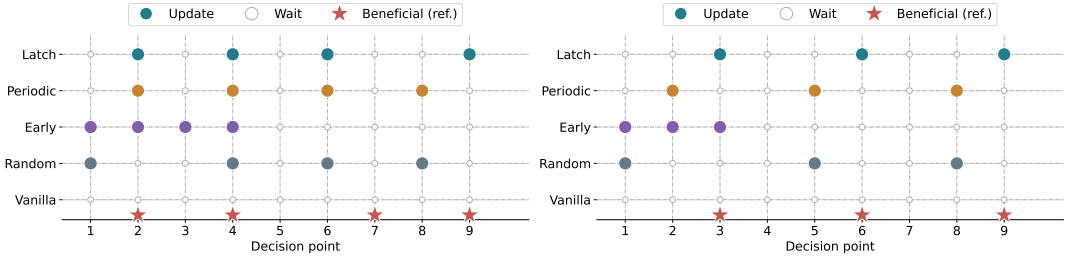
Figure 6: Ablation study across different representative reasoning tasks on LLaMA-3.1-8B.

largest degradation on GSM8K and BBH, indicating the value of additional latent reasoning for identifying reusable corrections. This result suggests that historical trajectories and observed outcomes alone provide incomplete evidence about what the agent could gain from adaptation. Removing realizability analysis has the greatest impact on MBPP+, highlighting the importance of assessing whether query-specific corrections can be jointly internalized through a shared parameter update. The observed decline supports distinguishing corrections revealed by deeper reasoning from those that the reasoning adapter can effectively realize. On MuSiQue, removing advantage prediction causes the largest decline, supporting the role of future performance and capability retention in selecting update opportunities. This finding emphasizes the importance of evaluating the anticipated consequences of adaptation alongside the correction evidence available at the current decision point. Removing latent distillation also reduces performance across all four benchmarks, showing that terminal-state alignment contributes beyond behavioral distillation and replay. Although this ablation retains more performance than the other variants, its consistent deficit supports explicitly transferring deeper latent representations into the shallow policy. Together, these results support the complementary roles of correction discovery, realizability analysis, advantage prediction, and latent distillation in determining when to evolve and how to realize the selected updates.



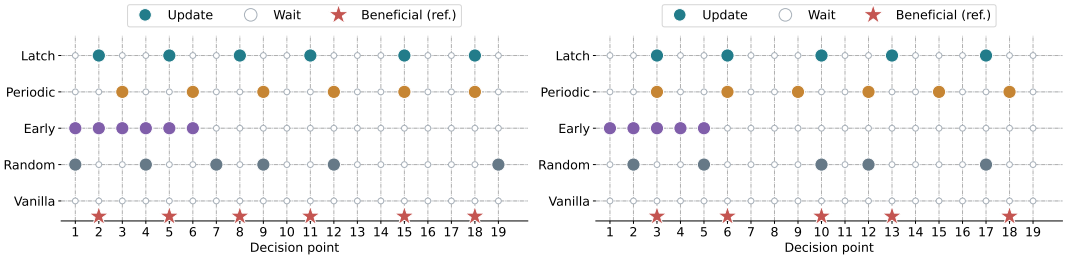
(a) LLaMA-3.1-8B (b) Qwen3.8-27B

Figure 7: Evolution timing on MATH with different backbones.



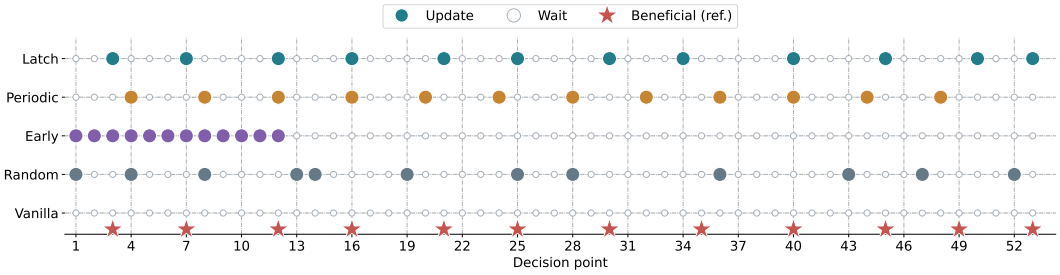
(a) LLaMA-3.1-8B (b) Qwen3.8-27B

Figure 8: Evolution timing on HumanEval+ with different backbones.

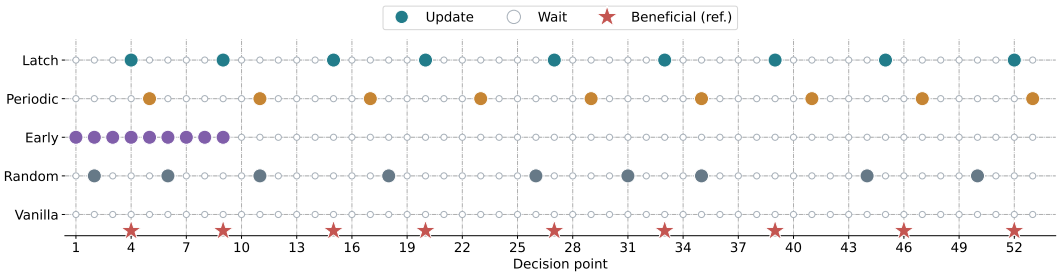


(a) LLaMA-3.1-8B (b) Qwen3.8-27B

Figure 9: Evolution timing on MBPP+ with different backbones.



(a) LLaMA-3.1-8B



(b) Qwen3.8-27B

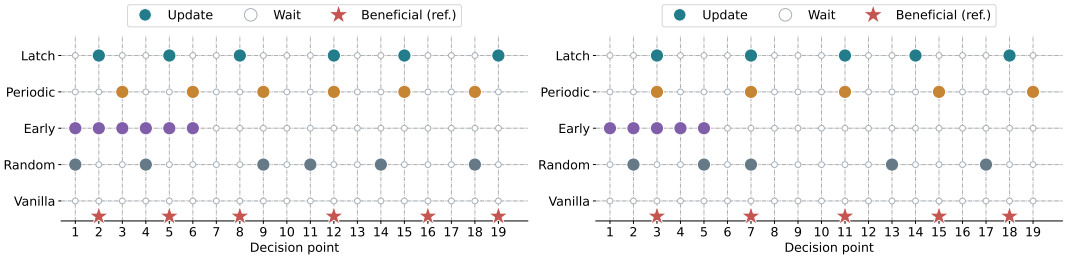
Figure 10: Evolution timing on BBH with different backbones.

E.2 Additional Evolution Timing Analysis

Figures 7–12 extend the evolution-timing analysis to MATH, HumanEval+, MBPP+, BBH, HotpotQA, and MuSiQue under both backbones. Red stars identify decision points with positive paired advantage along the fixed reference trajectory for each backbone. These markers provide a common reference for comparing the update schedules within each panel. Across the six benchmarks, LATCH generally exhibits closer alignment with the marked points than the predefined timing policies, while distributing its updates across different stages of the stream.

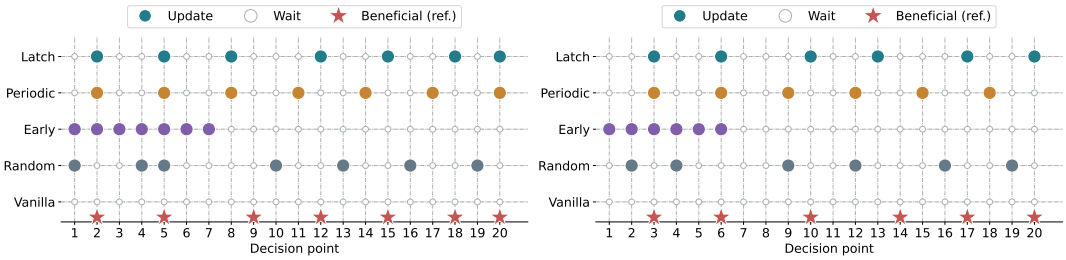
The code-generation benchmarks illustrate this distinction at matched update counts. On MBPP+ with LLaMA-3.1-8B, LATCH and PERIODIC each perform six updates, but all six LATCH updates coincide with marked points, compared with two periodic updates. On HumanEval+ with Qwen3.8-27B, LATCH selects all three marked points, whereas the three periodic updates fall between them. Thus, the difference in reference alignment arises from the selected positions even when the number of updates is identical. The HumanEval+ result with LLaMA-3.1-8B also shows that this alignment is not uniformly exact: LATCH updates at point 6, while the corresponding marked opportunity occurs at point 7.

On MATH, LATCH selects updates throughout the stream and captures several marked opportunities beyond the initial batches. With LLaMA-3.1-8B, five of its seven updates coincide with reference



(a) LLaMA-3.1-8B (b) Qwen3.8-27B

Figure 11: Evolution timing on HotpotQA with different backbones.



(a) LLaMA-3.1-8B (b) Qwen3.8-27B

Figure 12: Evolution timing on MuSiQue with different backbones.

markers; with Qwen3.8-27B, five of its six updates coincide. In comparison, EARLY allocates its updates entirely to the beginning of the stream, leaving subsequent marked opportunities unused. The periodic schedule covers a broader temporal range, but its fixed spacing often places updates away from the reference markers.

The longer BBH streams make these scheduling differences more apparent. LATCH continues to select updates across early, middle, and late stages, maintaining substantial overlap with the reference points over the extended sequence. EARLY exhausts its updates near the beginning, while the periodic and random schedules show less consistent alignment with the marked opportunities. The BBH results therefore complement the shorter streams by showing that the reference alignment of LATCH persists across a larger number of decision points.

The multi-hop question-answering benchmarks exhibit a similar pattern. On HotpotQA, five of the six LATCH updates with LLaMA-3.1-8B and four of the five updates with Qwen3.8-27B coincide with reference markers. On MuSiQue, the corresponding counts are six of seven and five of six. The remaining differences are localized: for example, LATCH updates at point 8 rather than the marked point 9 on MuSiQue with LLaMA-3.1-8B, and at point 13 rather than point 14 with Qwen3.8-27B. Across both datasets, the selected schedules nevertheless capture marked opportunities at multiple stages of the stream.

Finally, the schedules vary across backbones. In the displayed streams, LATCH performs fewer updates with Qwen3.8-27B than with LLaMA-3.1-8B on all six benchmarks. The selected positions also change: on HumanEval+, for instance, the schedule shifts from points $\{2, 4, 6, 9\}$ to $\{3, 6, 9\}$. These differences illustrate that the learned evolution policy produces schedules specific to the current agent, adapting both update frequency and placement to its decision-point evidence.

E.3 ADDITIONAL HYPERPARAMETER SENSITIVITY ANALYSIS

Figures 13 and 14 report additional sensitivity results on GSM8K and HumanEval+ under both backbones. Across the tested configurations, both benchmarks favor $H = 2$, $N_T = 4$, $d = 4$, and $|\mathcal{P}_t| = 16$, consistent with the settings used in our main experiments.

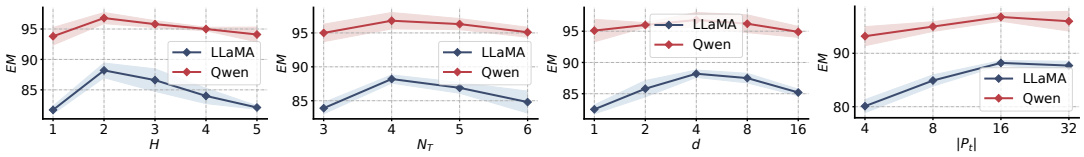


Figure 13: Hyperparameter analysis on GSM8K under various backbones.

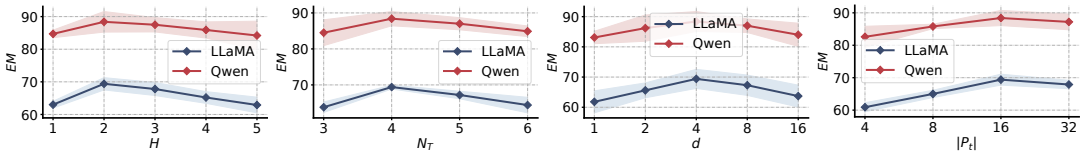


Figure 14: Hyperparameter analysis on HumanEval+ under various backbones.

Increasing the prediction horizon from $H = 1$ to $H = 2$ improves performance on both benchmarks, while longer horizons progressively reduce it. This pattern suggests that the immediate next batch provides an incomplete view of an update's value, whereas a moderate horizon captures useful downstream effects. The decline at larger H is consistent with the greater difficulty of predicting outcomes across additional changes in the stream. LLaMA exhibits a larger variation across horizons, particularly on GSM8K, indicating greater sensitivity to this choice than Qwen in the evaluated settings.

For the deep reasoning path, increasing N_T from 3 to 4 improves performance, but further increases provide no additional benefit. The peak at $N_T = 4$ suggests that additional latent reasoning supplies useful correction signals up to a moderate depth. Beyond that point, deeper targets may become harder for the fixed shallow path to approximate and internalize. The similar trend on mathematical reasoning and code generation supports selecting the deep-path length according to the usefulness of its corrections rather than reasoning depth alone.

The subspace dimension exhibits a comparable pattern. Performance improves as d increases from 1 to 4, then declines at $d = 8$ and $d = 16$. A very small subspace restricts the parameter directions available to explain the observed corrections, while a larger subspace does not necessarily provide a more informative realizability signal. The results favor a compact set of shared directions, with $d = 4$ providing the strongest performance across both benchmarks and backbones.

Finally, increasing the diagnostic set size from 4 to 16 consistently improves performance. This is consistent with broader query coverage supporting the estimation of shared corrections and their cross-query consistency. Expanding the set to 32 yields no further improvement and produces a small decline in the plotted means. Thus, $|\mathcal{P}_t| = 16$ provides sufficient diagnostic evidence in these experiments, while the additional queries offer diminishing returns. Together, these results support the selected configuration across both mathematical reasoning and code generation.