SLEEP-TIME TRAINING: FROM DEPLOYMENT EXPERIENCE TO PERSISTENT AGENT IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale pre-training and post-training produce capable LLM agents, but their parameters typically remain static after deployment. Existing deployment-time improvement methods mainly rely on additional computation during user interaction or external memory, which either increases user-facing latency or does not directly improve the agent's policy. We introduce *Sleep-Time Training* (STT), a deployment-time learning paradigm in which agents improve during user-idle periods. During user-facing sessions, the agent collects interaction trajectories. During sleep time, an auxiliary model converts these trajectories into corrected demonstrations and identifies defective intermediate actions. We first learn from replayed demonstrations with supervised fine-tuning, and then perform group-relative reinforcement learning over original and iteratively repaired trajectories. To study persistent improvement across repeated interactions, we introduce STT-Bench, a four-day benchmark containing 20 multi-day base task families and 20 held-out generalization families, with hidden user preferences and different levels of preference disclosure. Experiments show that STT improves future task completion and preference adherence without increasing user-facing inference cost, while retaining gains across multiple days and transferring to related unseen tasks.

1 INTRODUCTION

Large language model (LLM) agents are rapidly evolving from dialogue systems into interactive systems that can use tools, write code, browse the web, and assist users in long-horizon tasks (OpenAI, 2026; Anthropic, 2026; Qwen Team, 2026). Personal agents such as OpenClaw (OpenClaw, 2026), Codex (OpenAI, 2025), and Claude Code (Anthropic, 2025) demonstrate that modern agents can execute increasingly complex workflows. In the natural deployment setting of personal agents, a user may ask an agent to organize files, debug a codebase, book travel, or maintain a project; the agent acts through tools, receives corrections, sometimes fails, and the user leaves. Hours or days later, the user returns with related goals, unfinished workflows, and recurring latent preferences. This repeated-session lifecycle makes deployment itself a rich source of experience for learning (Silver & Sutton, 2025).

Despite this rich stream of experience, agents are largely static between different sessions in current deployment paradigms (OpenClaw, 2026). Their parameters are fixed after large-scale pre-training and post-training, even as they observe user corrections, implicit preferences, tools failures, and successful strategies. As a result, an agent may spend one session discovering how a user wants reports formatted or which coding style a project follows, only to require similar guidance again in the next session, or merely store this in external memory (in a *passive* manner) without internalizing it into its policy. This exposes a central mismatch: personal agents are deployed in evolving, user-specific workflows, but their policies typically do not evolve after deployment.

A natural question therefore arises: *how should personal agents continue to improve during deployment?* Existing approaches mostly improve agents along two axes. First, test-time scaling allocates more computation to the current query while the user is waiting, for example by sampling more reasoning paths and selecting the best one (Snell et al., 2025; Liu et al., 2025). This can improve the current response, but it increases user-facing latency and produces only temporary improvement. Second, external memory-based methods store facts or strategies and retrieves them for later tasks (Chhikara et al., 2025; Ouyang et al., 2026; Zhang et al., 2026a;b). Although useful, the improvement remains external to the policy and depends on successful retrieval, and real user interactions are often sparse, noisy, fragmented, and not immediately suitable as dense training data.

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.

In particular, personal-agent trajectories usually contain incomplete attempts, implicit preferences, partial corrections, tool failures, abandoned subtasks, and delayed success signals. Across existing paradigms, a critical missing mechanism is how we can transform naturally occurring deployment trajectories into reusable learning experiences for persistent policy improvement.

In this work, we introduce *Sleep-Time Training* (STT), a continuous deployment-time learning paradigm for personal agents. The key observation is that personal agents naturally have a recurring "sleep" period: users interact with the agent during "day-time", while the agent is often unused during "sleep-time" (e.g., overnight or between task sessions). The central idea is to make the repeated-session structure of personal-agent use part of the learning algorithm to continuously evolve the agent's internalized policy, which is largely overlooked in current paradigms. During day-time, the agent accumulates trajectories from real use, and then converts them into learning signals and updates itself before the user returns during sleep-time. Real deployment trajectories, however, cannot be naively repurposed for training. They often suffer from cascading tool-use failures, unnecessary actions, and sparse outcome signals. Moreover, when the current agent rarely completes a task successfully, standard outcome-based RL (Shao et al., 2024; Yu et al., 2025) collapses due to limited effective learning signals. The key challenge is therefore how to transform sparse, imperfect traces into reliable and sufficiently dense supervision for persistent policy improvement.

To resolve this challenge, we develop a novel two-stage sleep-time training procedure. First, an auxiliary model reconstructs imperfect deployment trajectories into corrected demonstrations, which are replayed and validated in the environment before being incorporated into supervised fine-tuning. It rapidly recalibrates the agent's action priors towards viable execution paths for supervision. Second, we perform iterative step repair for reinforcement learning. STT identifies the earliest defective agent action and replaces only this action, after which the agent continues the trajectory by itself. We then perform group-relative RL by pairing the original trajectories and repaired counterparts. This design provides dense supervision from corrected demonstrations while retaining the agent's own interaction process during reinforcement learning. Rather than remaining silent during this period, the agent can use sleep-time computation to consolidate past interactions, repair previous failures, synthesize new training environments, and update itself before the user returns. Unlike test-time scaling (which offloads computational overhead onto waiting users during live interaction), STT asks how to make the agent persistently stronger before the next query arrives. Furthermore, unlike sleep-time compute (Lin et al., 2025), which primarily pre-computes answers or intermediate reasoning for anticipated future queries, STT internalizes both general execution reliability and personalized user preferences into the agent's policy. Table 1 summarizes these distinctions in terms of user-facing latency, use of deployment experience, policy updates, and persistent gains.

Evaluating this setting requires benchmarks beyond single-session task success. A sleep-time training benchmark should test whether an agent improves after prior interactions, retains gains across days, infers hidden user preferences, and transfers experience to related unseen tasks. We therefore introduce STT-Bench, comprising 20 multi-day base task families, 20 held-out generalization families, and controlled levels of preference disclosure. Across multiple models, STT consistently outperforms memory-based and RL baselines. The gains persist across sleep cycles, transfer to held-out tasks, and remain substantial when the prompt no longer states the user's preferences, suggesting that deployment trajectories can be converted into reusable learning signals for continual agent improvement.

Our contributions are as follows:

- We introduce *Sleep-Time Training* (STT), a novel deployment-time learning paradigm that converts interaction experience into parameter updates during user-idle periods, enabling persistent agent improvement without increasing user-facing inference cost.
- We develop a two-stage sleep-time training method that transforms imperfect deployment trajectories into effective learning signals. We first learn from replayed corrected demonstrations with supervised fine-tuning, and then perform group-relative reinforcement learning over original and iteratively repaired trajectories.
- To evaluate the sleep-time training capabilities of personal agents, we introduce STT-Bench, a four-day benchmark with 20 multi-day base task families, 20 held-out generalization families, and controlled levels of preference disclosure. Experiments show that STT outperforms memory-based and RL baselines, retains improvements across days, and transfers to both future task instances and unseen task families.

Table 1: Comparison of deployment-time improvement paradigms. STT uniquely turns deployment experience into persistent policy improvement during user-idle time.

Paradigm	User Latency	Uses Deployment Experience	Updates Policy	Persistent Gain
Test-Time Scaling (Wang et al., 2023; Snell et al., 2025)	✗	✗	✗	✗
External Memory / Skills (Chhikara et al., 2025; Ouyang et al., 2026)	✓	✓	✗	✗
Sleep-Time Scaling (Lin et al., 2025)	✓	✗	✗	✗
Sleep-Time Training (Ours)	✓	✓	✓	✓

2 Related Work

Evolution of LLM Agents. RL has become an effective approach for improving the agentic capabilities of LLM agents (Zhang et al., 2025; Dong et al., 2026b). OpenClaw-RL (Wang et al., 2026b) trains claw agents using process rewards and feedback-conditioned distillation signals extracted from the next interaction state. MetaClaw (Xia et al., 2026) jointly evolves a reusable skill library and the agent policy, scheduling RL-based parameter updates during user-inactive periods. Claw-R1 (Wang et al., 2026a) provides step-level data infrastructure for collecting, evaluating, and curating agent interactions for RL training. Another line of work improves LLM agents from the data perspective (Ma et al., 2026; Dong et al., 2026a; Bai et al., 2026). SkillClaw (Ma et al., 2026) proposes to leverage multi-user interaction data to evolve a shared skill library. Agent-World (Dong et al., 2026a) proposes to synthesize environments for agentic RL training. ClawGym (Bai et al., 2026) proposes a framework that synthesizes the SFT data and RL environments for claw agent learning. Closely related to our trajectory repair, Agent-R (Yuan et al., 2025) constructs revision trajectories with Monte Carlo tree search for iterative self-training, while SDPO (Hübotter et al., 2026) distills feedback-conditioned predictions into the policy. Our method combines corrected demonstrations with group-relative learning from original trajectories and agent continuations after local repair.

Deployment-Time Improvement. Deployment-time improvement mainly includes test-time improvement (Snell et al., 2025; Liu et al., 2025; Zuo et al., 2025) and sleep-time improvement (Lin et al., 2025). Test-time scaling improves reasoning through additional inference computation with fixed model parameters, for example by sampling diverse reasoning paths and aggregating their answers through self-consistency (Wang et al., 2023). Compute-optimal scaling allocates this computation according to problem difficulty and the choice of policy and process reward models (Snell et al., 2025; Liu et al., 2025). Test-time training additionally updates model parameters using inference-time data. Hardt & Sun (2023) fine-tune language models on retrieved neighbors of each test input, while TTRL (Zuo et al., 2025) performs RL on unlabeled test queries using majority-vote-based reward estimates. Lin et al. (2025) proposes sleep-time scaling by processing available context before queries arrive, reducing subsequent inference computation. ProAct (Hu et al., 2026) uses a similar concept to anticipate user needs and augment agent memory during idle time. Test-time scaling can increase user-facing latency, while context-based sleep-time methods retain acquired knowledge in external memory rather than model parameters. Our method uses idle periods to internalize deployment experience through trajectory repair and policy training, without increasing the inference budget for subsequent interactions.

3 STT-Bench

Existing tool-use benchmarks primarily measure task completion within a single interaction (PinchBench, 2026; Ye et al., 2026; Ding et al., 2026), which is insufficient for evaluating our new sleep-time training paradigm: an agent may complete a task by following explicit instructions, yet fail to internalize user-specific preferences, transfer them across days, or improve when the next prompt omits those preferences. STT-Bench is designed to evaluate this missing capability.

Specifically, STT-Bench is built around three design principles. First, tasks span multiple chronological days to measure whether deployment experience leads to persistent improvement in later interactions. Second, we vary how explicitly user preferences are revealed in the current prompt to test whether the agent has learned these preferences from previous interactions. Third, we include held-out generalization task families to evaluate whether learned user-specific behavior transfers to related but unseen workflows.

Persona and Task Synthesis. We construct STT-Bench from realistic tool-use scenarios. We first collect documentation for 256 frequently used Model Context Protocol (MCP) servers from Smithery[1] and retain their server descriptions and tool definitions after data cleaning.

We use strong coding agents (Codex) to construct user personas, task families, mock services, and executable task instances. The persona describes recurring user preferences that can affect how a task should be completed, such as preferred communication style, formatting conventions, workflow choices, or constraints on external actions.

To separate memorization from transfer, STT-Bench contains 40 task families, split into 20 *base* families and 20 *generalization* families. Base families are available during sleep-time learning and evaluation. Generalization families are used only for evaluation. Each generalization family shares user-level habits or skills with the base split but requires a different task scenario or workflow, allowing us to test whether learned behavior transfers beyond previously observed tasks.

Each base family contains four chronological instances, one for each day, while each generalization family contains one held-out Day 4 instance, yielding 100 task instances in total. Every instance uses the same user persona. We additionally construct four prompt levels, summarized in Table 2, that vary how explicitly the prompt reveals the user's preferences. Higher levels therefore place more emphasis on learning user-specific behavior from earlier interactions rather than from the current prompt.

Table 2: Preference-disclosure levels in STT-Bench.

Level	Prompt disclosure
L0	All preferences are explicit.
L1	One preference is explicit.
L2	Prior habits are referenced indirectly.
L3	No preference hint is provided.

Evaluation Protocol. To enable reproducible and safe evaluation, all tasks are executed against mock services rather than live external services, following the general design of Claw-Eval (Ye et al., 2026). The agent can interact with the environment only through the declared tool interfaces.

We evaluate two complementary dimensions: *task completion* and *preference adherence*. Scores are computed from service-side audit logs and the final task artifact rather than from the final textual response alone. For a trajectory τ, the overall score is:

$$\text{Score}(\tau) = C(\tau) \cdot E(\tau) \cdot P(\tau), \tag{1}$$

where C measures task completion, E measures execution efficiency, and P measures preference adherence. Task completion is the weighted average of the task's completion checks, including correct entities, required state changes, and artifact quality. Preference adherence is the average score over the three user preferences. Efficiency penalizes unnecessary assistant steps relative to the task's reference workflow. Appendix C provides the detailed scoring rules.

This decomposition is important for Sleep-Time Training: an agent may learn how to complete a task while still failing to internalize user-specific preferences. We therefore report C, P, and the overall score separately throughout our experiments.

Task Validation. Task generation is followed by automatic execution and human verification. During synthesis, the coding agent executes the reference workflow against the mock service and checks whether the task, tools, and verifier are mutually consistent. Invalid tool schemas, unreachable states, ambiguous requirements, and inconsistent verifier conditions are repaired before the task is admitted to the benchmark.

We then perform two sanity checks on the evaluator. First, the reference trajectory must obtain an overall score greater than 0.9. Second, a null trajectory that performs no meaningful task action must obtain a score below 0.1. These checks detect overly permissive or incorrectly specified evaluators.

Finally, we manually inspect all benchmark families to verify that: (1) tasks are executable; (2) prompts do not leak hidden evaluation criteria; (3) evaluators align with the intended task objective; (4) user preferences are meaningful and consistently represented across days; and (5) base and generalization families do not share identical workflows. After filtering, STT-Bench comprises 40 human-verified task families. Appendix B details the construction and calibration pipeline.

[1] https://smithery.ai/

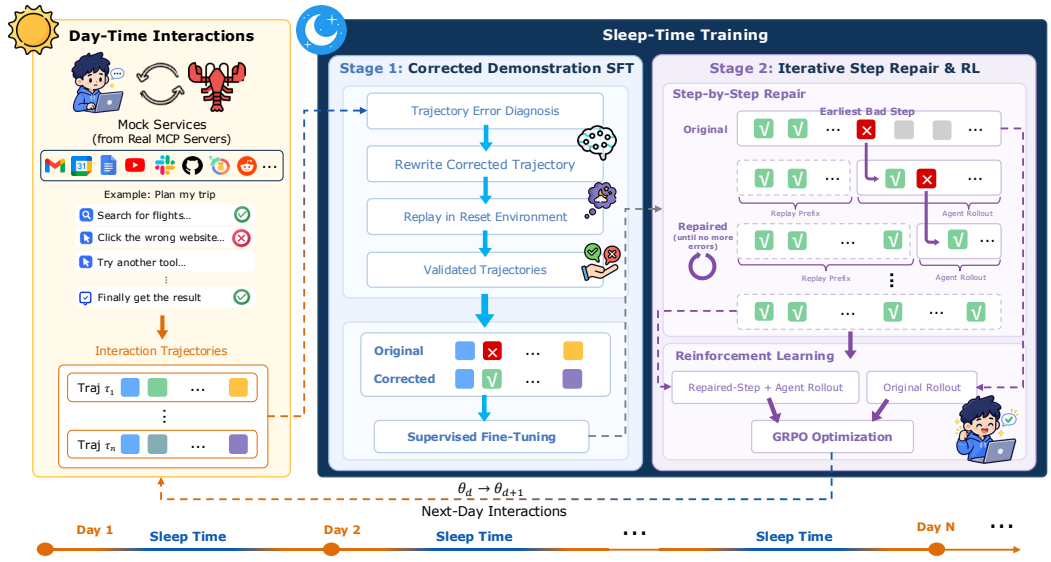


Figure 1: Overview of STT. The agent collects interaction trajectories during user-facing sessions and converts them into policy updates during sleep time. First, an auxiliary model reconstructs, replays, and validates corrected demonstrations for SFT. Second, iterative step repair pairs original trajectories with repaired continuations for group-relative RL.

4 Sleep-Time Training

4.1 Problem Setting

Consider an agent with policy π_{θ_d} at the beginning of day d. For a task x, the agent observes the interaction history h_t and produces an assistant action a_t; tool-use actions yield environment observations o_t. A complete trajectory is

$$\tau = (x, a_1, o_1, \ldots, a_{T-1}, o_{T-1}, a_T), \tag{2}$$

where the final action contains the user-facing response.

Let $\mathcal{H}_d$ denote the interaction experience collected on day d. During sleep time, the agent updates its parameters according to

$$\theta_{d+1} = U(\theta_d, \mathcal{H}_d; B_d), \tag{3}$$

where B_d is the available sleep-time computation budget. Let $\mathcal{D}_{\text{future}}$ denote the distribution of future tasks and $S(x, \tau)$ the performance score of trajectory τ on task x. Given B_d, STT seeks an update U that maximizes the future-task objective

$$J_{\text{STT}}(\theta_{d+1}) = \mathbb{E}_{\substack{x \sim \mathcal{D}_{\text{future}} \\ \tau \sim \pi_{\theta_{d+1}}(\cdot|x)}} \left[S(x, \tau) \right], \tag{4}$$

with θ_{d+1} defined by equation 3 and an unchanged user-facing inference budget. Future requests define this objective but are not available during training.

Our implementation updates on current-day trajectories only. During SFT, deterministic checks from the current training tasks provide additional feedback for demonstration repair, as detailed in Appendix C. RL repair and outcome scoring use only the public task and observed trajectory. Knowledge from earlier days is carried forward through the model parameters rather than by replaying earlier trajectories or appending them to future prompts.

4.2 Learning from Corrected Demonstrations

Reference collection. For each current-day task, we collect N independent trajectories with the current policy π_{θ_d}. Each attempt starts from the same initial task state. We select the first $M \leq N$ trajectories by sample index, without consulting benchmark scores. Each selected trajectory is the sole reference for one synthesized example. These additional attempts are part of the simulated experience-collection budget and must be counted when comparing methods.

Synthesis and replay. An auxiliary language model receives the public task, tool schemas, and one reference trajectory. It proposes a complete sequence of assistant actions that preserves useful content while correcting errors. The environment is then reset and executes these actions to produce actual observations. The auxiliary model never supplies tool results. If execution or review reveals a defect, the candidate is revised and replayed. For a model-diagnosed error, revisions preserve the assistant prefix before that error and rebuild the remaining sequence. Failed deterministic checks instead trigger full resynthesis with rule feedback. This stage uses auxiliary-generated complete trajectories; the agent does not generate their suffixes.

A candidate is accepted only after public-evidence review, deterministic training-task checks, and validity checks. It must contain a grounded final answer, legal tool calls, no unresolved execution errors, and fit the step and context budgets. Failed deterministic checks are returned to the auxiliary model for revision. Truncated or malformed candidates are excluded. These checks establish an executable demonstration, not proof that every semantic judgment is correct. The accepted demonstrations form $\mathcal{D}_{\mathrm{SFT}}$.

Supervised update. Writing each accepted demonstration as a context–output pair (c, y), we use the standard supervised objective

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{(c,y)\sim\mathcal{D}_{\mathrm{SFT}}}\left[\log \pi_\theta(y \mid c)\right]. \tag{5}$$

The likelihood is computed only over assistant tokens; user requests and tool observations are context. The resulting policy π_{SFT} initializes the RL stage.

4.3 Sleep-Time RL via Iterative Step Repair

For each RL update, we freeze the current agent as π_{old} and sample K new trajectories per task. The auxiliary model inspects each trajectory in order and identifies its earliest defective assistant step $t^\star$. A defect can be a wrong action, an unsupported statement, a missing deliverable, or clearly unnecessary repetition. The auxiliary model writes one replacement action $\widetilde{a}_{t^\star}$, using only information available before that step. Later observations can support diagnosis but cannot be inserted as prior knowledge into the replacement.

We reset the environment, replay the earlier actions, and execute $\widetilde{a}_{t^\star}$. The agent then generates a new suffix from the resulting history:

$$\widetilde{\tau} = \mathrm{Replay}(a_{<t^\star}) \oplus \widetilde{a}_{t^\star} \oplus \mathrm{Continue}_{\pi_{\mathrm{old}}}(\widetilde{h}_{t^\star+1}), \tag{6}$$

where $\oplus$ denotes concatenation and all intervening observations are obtained by execution. The auxiliary model reviews the full new trajectory again. This process continues until the trajectory is accepted. Reinspection can expose later failures or incorrect earlier judgments. The prefix and new suffix share the original step budget.

The final pair contains the original attempt and the last accepted repaired version. Intermediate revisions are not separate group members. An already acceptable original remains unchanged, so its pair can contain two identical trajectories. If repair cannot produce an accepted version within the budget, both members of that pair are excluded from the learning group. Other valid pairs for the same task remain usable.

Public outcome reward. A separate scoring request evaluates each original or repaired trajectory against the public task and its actual observations. It returns a binary reward $R_i \in \{0, 1\}$, where one indicates that all public requirements are correctly completed. Scoring requests do not receive repair decisions or the paired trajectory. A repair is therefore not assigned a positive reward by construction. A successfully recovered earlier error or inefficiency alone does not imply a zero reward. This criterion differs from the stricter acceptance criterion used to select training demonstrations.

The scorer receives neither hidden preferences nor the benchmark evaluator's rubric, private state, or scores. The same auxiliary model may serve repair and scoring through separate requests; this separation does not remove shared-model bias. The benchmark score in Section 3 is used only for evaluation.

Group Relative Policy Optimization (GRPO). For each task, the $0 < G \leq 2K$ retained original and repaired trajectories form a group $\{\tau_i\}_{i=1}^{G}$. Following GRPO (Shao et al., 2024), we compute the normalized advantage $\widehat{A}_i = (R_i - \bar{R})/\sigma_R$, where $\bar{R}$ and σ_R are the mean and standard deviation of

rewards in the group. Every assistant token in trajectory i shares $\widehat{A}_i$, and groups with identical rewards are skipped. Let $y_{i,j}$ and $h_{i,j}$ denote the j-th assistant token and its preceding history, and let L_i be the number of assistant tokens in τ_i. The importance ratio is $r_{i,j}(\theta) = \pi_\theta(y_{i,j} \mid h_{i,j})/\pi_{\text{old}}(y_{i,j} \mid h_{i,j})$. We minimize the standard clipped objective:

$$\mathcal{L}_{\text{RL}}(\theta) = -\frac{1}{G}\sum_{i=1}^{G}\frac{1}{L_i}\sum_{j=1}^{L_i}\min\Big(r_{i,j}(\theta)\widehat{A}_i,\ \text{clip}(r_{i,j}(\theta), 1-\varepsilon_{\text{low}}, 1+\varepsilon_{\text{high}})\,\widehat{A}_i\Big), \tag{7}$$

This within-trajectory average prevents longer trajectories from receiving more weight merely because they contain more tokens.

All retained assistant tokens participate, including auxiliary-written replacement actions. Both probabilities in $r_{i,j}$ are evaluated by the agent on the stored token sequence. Because repaired trajectories mix agent generation with external correction, the objective in equation 7 is GRPO-style training on mixed data rather than an unbiased on-policy policy-gradient estimator. This distinction motivates the teacher-token and repair ablations in Section 5.3.

Why combine the two stages? The two stages address different failure regimes. Early in evolution, the agent may rarely produce a fully successful trajectory, making outcome-based RL nearly signal-free. Corrected demonstrations provide dense supervision in this regime. After SFT, iterative repair exposes the agent to states induced by its own actions and creates local contrasts between unsuccessful and improved trajectories. The agent must then continue the task by itself after the repaired step. This prevents sleep-time learning from degenerating into pure imitation of complete auxiliary-model solutions. The complete multi-day procedure is summarized in Algorithm 1 in Appendix A.

5 Experiments

5.1 Setup

Models and Baselines. We conduct experiments using Qwen3.5-4B, Qwen3.5-9B, and Qwen3-30B-A3B-Instruct-2507 as the base models (Yang et al., 2025; Qwen Team, 2026). We use non-thinking mode for all three models. We compare STT against several representative baselines: (1) Base Model, (2) ReasoningBank (Ouyang et al., 2026), (3) MemEvolve (Zhang et al., 2026a), and (4) OpenClaw-RL (Wang et al., 2026b). Details of these baselines are provided in Appendix C.

Training Details. We implement STT with slime v0.3.0 (Zhu et al., 2025) and use SGLang (Zheng et al., 2024) for LLM inference. We train on the 20 base task families at L1 for three days, using 3, 2, and 1 SFT epochs on Days 1–3, respectively, followed by 4 RL steps per day. Each day starts from the previous day's final checkpoint; no checkpoint selection is performed. The SFT and RL learning rates are 1.0×10^{-5} and 1.0×10^{-6}, respectively. We use GPT-5.6-Luna for synthesis, repair, and scoring, and run each experiment on 8 NVIDIA H800 GPUs. Additional implementation details are provided in Appendix C.

Evaluation Details. We evaluate on both the base and generalization splits of STT-Bench at L1–L3. Day 4 tasks and all 20 generalization families are excluded from learning. We evaluate earlier days to measure retention and Day 4 to measure transfer. For each task and prompt level, we sample four independent trajectories at a temperature of 0.7 and report the average score (avg@4). Each trajectory has at most 32 assistant steps, with model parameters and memories fixed during evaluation. Additional evaluation details are provided in Appendix C.

5.2 Main Results

STT outperforms the baseline methods. Table 3 shows that STT consistently outperforms the memory-based methods and OpenClaw-RL across all three backbones. On Qwen3.5-4B, it obtains an overall score of 40.8, exceeding the strongest baseline, OpenClaw-RL, by 19.0 points (40.8 vs. 21.8). On Qwen3.5-9B, the corresponding improvement is 9.4 points (41.0 vs. 31.6). For seen tasks, STT improves over the strongest baseline by 22.6 and 12.7 points on the 4B and 9B models, respectively. Notably, STT with the 4B model also exceeds every reported 9B baseline in overall score by at least 9.2 points, indicating that its gains cannot be explained by model scale alone. On Qwen3-30B-A3B-Instruct-2507, STT reaches 20.2 overall, outperforming the strongest baseline by 6.2 points; its gains are 8.2 points on seen tasks and 3.5 points on unseen tasks. Although absolute scores are lower for this backbone across all methods, the consistent improvement shows that STT is not limited to the two dense Qwen3.5 models.

Table 3: Evaluation results on STT-Bench. The results of STT are shaded and the highest values are **bolded**.

Method	Seen Tasks				Unseen Tasks			Avg.
	Day 1	Day 2	Day 3	Avg.	Day 4 Base	Day 4 Gen.	Avg.	
Qwen3.5-4B								
↳ ReasoningBank	24.4	22.3	28.3	25.0	22.8	10.8	16.8	20.9
↳ MemEvolve	25.0	25.0	25.4	25.1	20.9	11.1	16.0	20.6
↳ OpenClaw-RL	25.9	25.4	25.3	25.5	23.5	12.8	18.2	21.8
↳ STT (Ours)	**45.4**	**48.4**	**50.6**	**48.1**	**40.3**	**26.4**	**33.4**	**40.8**
Qwen3.5-9B								
↳ ReasoningBank	33.5	33.1	33.4	33.3	34.2	19.1	26.6	30.0
↳ MemEvolve	35.0	33.8	31.6	33.4	34.1	19.4	26.8	30.1
↳ OpenClaw-RL	36.2	37.2	32.6	35.3	34.8	21.0	27.9	31.6
↳ STT (Ours)	**46.0**	**47.2**	**50.8**	**48.0**	**41.5**	**26.6**	**34.0**	**41.0**
Qwen3-30B-A3B-Instruct-2507								
↳ ReasoningBank	14.4	13.9	14.8	14.4	15.0	12.2	13.6	14.0
↳ MemEvolve	17.4	13.2	12.8	14.5	14.4	11.7	13.1	13.8
↳ OpenClaw-RL	13.8	15.2	16.4	15.1	13.2	11.2	12.2	13.7
↳ STT (Ours)	**22.3**	**23.7**	**23.9**	**23.3**	**18.8**	**15.4**	**17.1**	**20.2**

STT generalizes to unseen tasks. The gains from STT extend beyond the tasks used for sleep-time learning: Day 4 instances and all generalization families are excluded from training. On Qwen3.5-4B, STT scores 40.3 on held-out Day 4 instances from the base families and 26.4 on unseen generalization families, outperforming the strongest corresponding baselines by 16.8 and 13.6 points. On Qwen3.5-9B, it scores 41.5 and 26.6 on these two splits, with gains of 6.7 and 5.6 points. On Qwen3-30B-A3B-Instruct-2507, it scores 18.8 and 15.4, exceeding the strongest baselines by 3.8 and 3.2 points. Averaged across both unseen splits, STT improves the strongest baseline from 18.2 to 33.4 on Qwen3.5-4B, from 27.9 to 34.0 on Qwen3.5-9B, and from 13.6 to 17.1 on Qwen3-30B-A3B-Instruct-2507. Thus, STT transfers learned behavior to both new instances from familiar task families and related workflows that were never observed during training.

STT continuously improves across multiple days. Figure 2 tracks Qwen3.5-4B transfer to held-out Day 4 tasks after the SFT and RL stages of each daily sleep-time update. The combined Day 4 Base+Gen. score rises across all three cycles and reaches 33.4 after Day 3. In contrast, ReasoningBank, MemEvolve, and OpenClaw-RL end at 16.8, 16.0, and 18.2, respectively. The same pattern holds for both transfer settings: STT reaches 40.3 on new instances of the base families and 26.4 on unseen generalization families. Moreover, at the final checkpoint, performance on the earlier Day 1 and Day 2 instances remains high at 45.4 and 48.4, alongside 50.6 on Day 3. These results show cumulative transfer while maintaining strong performance on earlier tasks.

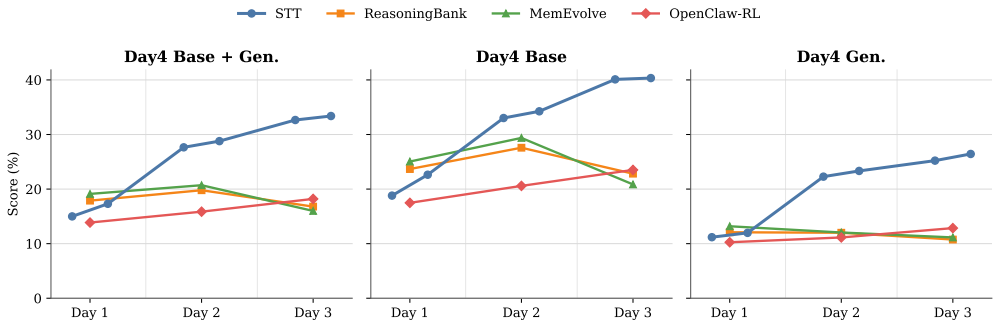


Figure 2: Multi-day performance on held-out Day 4 tasks. STT reports the checkpoints after SFT and after RL on each day and baselines report their daily checkpoints.

5.3 Analysis

Which learning stage matters? Table 4 compares STT with SFT-only and RL-only variants using Qwen3.5-4B; each starts from the same checkpoint and follows the same three-day task schedule and per-stage settings. The full method reaches 40.8 overall, outperforming SFT-only and RL-only by 16.5 and 17.1 points, with similarly large gains on seen and unseen tasks. SFT-only supplies complete corrected demonstrations but does not train recovery from agent-induced states, whereas

RL-only starts from a weak policy and receives limited signal from sparse binary outcomes. Their combination first establishes viable action priors and then improves recovery under the agent's own interaction process.

Table 4: Ablation of the two sleep-time learning stages on Qwen3.5-4B after three days. The highest values are **bolded**.

Method	Seen Tasks				Unseen Tasks			Avg.
	Day 1	Day 2	Day 3	Avg.	Day 4 Base	Day 4 Gen.	Avg.	
STT w/o RL (SFT only)	26.0	30.1	28.6	28.2	25.0	15.9	20.4	24.3
STT w/o SFT (RL only)	22.9	28.3	28.5	26.6	28.0	13.5	20.8	23.7
STT	**45.4**	**48.4**	**50.6**	**48.1**	**40.3**	**26.4**	**33.4**	**40.8**

Does STT internalize user preferences? Table 5 compares the final Qwen3.5-4B checkpoints as preference cues are reduced. L1 states one current preference explicitly, L2 only refers to the user's prior habits, and L3 provides no preference cue. STT scores 38.2 at L2 and 39.3 at L3, outperforming the strongest corresponding baselines by 16.3 and 15.8 points. Its L3 score retains 96.3% of its L1 performance despite receiving no preference hint. This robustness indicates that task-solving behavior and user preferences are carried by the updated policy rather than recovered only by following an explicit instruction.

Table 5: Overall STT-Bench scores of Qwen3.5-4B under different preference-disclosure levels after three days. The L1 results are the corresponding overall scores from Table 3.

Method	L1: One Explicit	L2: Implicit	L3: Hidden
ReasoningBank	20.9	21.9	23.5
MemEvolve	20.6	20.3	20.6
OpenClaw-RL	21.8	20.5	20.2
STT (Ours)	**40.8**	**38.2**	**39.3**

Efficiency of STT. Table 6 reports the offline training cost of STT. The 4B, 9B, and 30B models consume an average of 12.3, 13.7, and 20.9 H800 GPU-hours per day, respectively. This computation includes both SFT and RL but occurs entirely during sleep time, so the improved policy uses the same user-facing inference procedure and budget as the original model.

Table 6: Consumed GPU hours of STT for each day.

Model	Day 1	Day 2	Day 3	Avg.
Qwen3.5-4B (8 GPUs)	12.9	11.7	12.2	12.3
Qwen3.5-9B (8 GPUs)	13.1	14.0	14.2	13.7
Qwen3-30B-A3B-Instruct-2507 (8 GPUs)	20.7	21.2	20.8	20.9

6 Conclusion

We introduced Sleep-Time Training, a deployment-time learning paradigm in which LLM agents improve from interaction experience during user-idle periods. Our instantiation combines replayed corrected demonstrations with iterative step repair and group-relative reinforcement learning, and STT-Bench evaluates the resulting behavior across days, task families, and levels of preference disclosure. Across the 4B, 9B, and 30B backbones, STT improves the overall score over the strongest baselines by 19.0, 9.4, and 6.2 points, respectively, with consistent gains on held-out Day 4 tasks. On Qwen3.5-4B, it also retains a score of 39.3 when the prompt provides no preference hint, compared with 23.5 for the strongest baseline. Removing either SFT or RL reduces the overall score from 40.8 to 24.3 or 23.7, confirming that dense corrected demonstrations and repair-based policy learning are complementary. These gains are obtained through offline sleep-time updates without changing the user-facing inference budget. More broadly, the results show that carefully repaired and verified deployment experience can produce persistent policy improvement rather than remaining only transient context or external memory.

Limitations. Our study has several limitations. Our benchmark covers fictional persona and four days simulation, with simulated tools rather than live services. For tasks that are extremely difficult and far above the capabilities of the current agent or strong proprietary LLMs, STT still cannot resolve them.

AI USE STATEMENT

In this work, we used generative AI tools for generating synthetic data sets. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, assisting with translation, cleaning and reformatting datasets, supporting qualitative and thematic data analysis, or interpreting results. Additionally, we used generative AI tools for editing a research paper to improve readability. We have reviewed all AI-assisted work. We validated the quality of LLM-generated dataset and checked the polished content. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The benchmark represents a fictional user and fictional private records, together with retained public source material. Task actions run in simulated environments and do not send real messages, make purchases, or alter external accounts.

REPRODUCIBILITY STATEMENT

We have provided the implementation details in Section 5 and Appendix C. To ensure the reproducibility of this work, we will release our code and data upon paper acceptance.

REFERENCES

Anthropic. Claude code: An agentic coding tool. https://docs.anthropic.com/en/docs/claude-code/, 2025. Accessed: 2026-06-30.

Anthropic. Claude fable 5 and claude mythos 5. https://www.anthropic.com/news/claude-fable-5-mythos-5/, 2026. Accessed: 2026-06-30.

Fei Bai, Huatong Song, Shuang Sun, Daixuan Cheng, Yike Yang, Chuan Hao, Renyuan Li, Feng Chang, Yuan Wei, Ran Tao, et al. Clawgym: A scalable framework for building effective claw agents. *arXiv preprint arXiv:2604.26904*, 2026.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Yang JingYi, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, et al. Wildclawbench: A benchmark for real-world, long-horizon agent evaluation. *arXiv preprint arXiv:2605.10912*, 2026.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026a.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic reinforced policy optimization. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=TX4k7BF6aO.

Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. *arXiv preprint arXiv:2305.18466*, 2023. URL https://arxiv.org/abs/2305.18466.

Haoyi Hu, Qirong Lyu, Xianghan Kong, Weiwen Liu, Jianghao Lin, Zixuan Guo, Yan Xu, Yasheng Wang, Weinan Zhang, and Yong Yu. Anticipate and learn: Unleashing idle-time compute in proactive agents. *arXiv preprint arXiv:2605.25971*, 2026.

Jonas Hübotter, Frederike Lübeck, Lejs Deen Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=QkfkxyRizZ.

Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E Gonzalez. Sleep-time compute: Beyond inference scaling at test-time. *arXiv preprint arXiv:2504.13171*, 2025.

Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *arXiv preprint arXiv:2502.06703*, 2025.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. Skillclaw: Let skills evolve collectively with agentic evolver. *arXiv preprint arXiv:2604.08377*, 2026.

OpenAI. Codex: A cloud-based software engineering agent. https://openai.com/index/introducing-codex/, 2025. Accessed: 2026-06-30.

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6/, 2026. Accessed: 2026-09-20.

OpenClaw. OpenClaw. https://github.com/openclaw/openclaw/, 2026. GitHub repository. Accessed: 2026-06-30.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/980ea04d23d1f6908964eba2a74afe45-Paper-Conference.pdf.

PinchBench. PinchBench: Real-world benchmarks for ai coding agents. https://github.com/pinchbench/skill/, 2026. GitHub repository. Accessed: 2026-06-30.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1(11):6, 2025.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Daoyu Wang, Mingyue Cheng, Qingchuan Li, Shuo Yu, Jie Ouyang, and Qi Liu. Claw-r1: A step-level data middleware system for agentic reinforcement learning. *arXiv preprint arXiv:2606.09138*, 2026a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

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

Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, Haonian Ji, Yuyin Zhou, et al. Metaclaw: Just talk–an agent that meta-learns and evolves in the wild. *arXiv preprint arXiv:2603.17187*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, et al. Claw-eval: Towards trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=2a36EMSSTp.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*, 2025.

Guibin Zhang, Haotian Ren, Chong Zhan, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng YAN. Memevolve: Meta-evolution of agent memory systems. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=qpkG0eKx4v.

Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, Yu Fu, Xingtai Lv, Yuchen Zhang, Sihang Zeng, Shang Qu, Haozhan Li, Shijie Wang, Yuru Wang, Xinwei Long, Fangfu Liu, Xiang Xu, Jiaze Ma, Xuekai Zhu, Ermo Hua, Yihao Liu, Zonglin Li, Huayu Chen, Xiaoye Qu, Yafu Li, Weize Chen, Zhenzhao Yuan, Junqi Gao, Dong Li, Zhiyuan Ma, Ganqu Cui, Zhiyuan Liu, Biqing Qi, Ning Ding, and Bowen Zhou. A survey of reinforcement learning for large reasoning models. *arXiv preprint arXiv:2509.08827*, 2025.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/8a94ff6f922d995d7d3f4ebf4143e442-Paper-Conference.pdf.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv. Accessed: 2026-06-30.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. TTRL: Test-time reinforcement learning. In *The Thirty-ninth Annual Conference on*

Neural Information Processing Systems, 2025. URL https://openreview.net/forum?id=VuVhgEiu20.

A Full Algorithm

Algorithm 1 Sleep-Time Evolving over D days

Input: Agent π_{θ_1}; public daily tasks $\mathcal{X}_{1:D}$; auxiliary model
Input: Attempts per task N, synthesized examples $M \leq N$, originals per group K, RL updates N_{RL}

```
for d = 1, ..., D do
    Run π_{θ_d} on each x ∈ X_d for N independent attempts.
    Correct the first M attempts per task separately; replay and review each candidate.
    Form D_SFT from accepted demonstrations and apply equation 5.
    for u = 1, ..., N_RL do
        Freeze the current agent as π_old.
        for each task x ∈ X_d do
            Sample K originals with π_old in reset environments.
            Iteratively repair each original and let π_old continue after each edit.
            Keep valid original–repaired pairs; score each member independently.
            Compute group-normalized advantages.
        end for
        Update with equation 7 if any group has nonzero advantage.
    end for
    Set θ_{d+1} to the final agent parameters.
end for
```

Output: Final policy $\pi_{\theta_{D+1}}$; no auxiliary model required for inference

B STT-Bench Construction Details

Instance organization. STT-Bench contains 100 stateful task instances derived from 40 task families. Families 001–020 form the base split and each contributes a distinct instance on Days 1–4, for 80 instances. Families 021–040 form the generalization split and each contributes one held-out Day 4 instance. The four instances of a base family retain the same three user preferences while changing the underlying business event, records, dates, and requested deliverable. Generalization family i is paired with base family $i - 20$ and shares at least one user preference, but uses a different scenario or workflow. All instances use the same fictional PrismForge founder persona.

Event-grounded environment construction. We construct each instance by first specifying a business event and then materializing the records that this event leaves across the relevant systems. Depending on the task, these records include email threads, documents, spreadsheets, calendars, task and issue trackers, code repositories, research sources, travel records, and financial data. The resulting benchmark contains 1,389 task-local environment records. Public source material is retained with provenance, whereas private company, personal, travel, and future-event records are synthetic; synthetic contacts use reserved `.test` addresses.

Each instance is stored as four artifacts: `config.yaml` contains the prompts and exposed tool schemas, `data.json` contains the initial mock-service state, `source.json` records provenance for the corresponding materials, and `grader.py` defines the executable evaluation specification. All reads and mutations go through the same stateful local tool service used by the agent and the reference policy. Thus, evaluation can verify actual document writes, drafts, calendar events, issue updates, and other state changes without contacting live services.

Preference-disclosure construction. Each family has three preferences, denoted P_1–P_3. We derive a neutral request by removing only preference-specific clauses from the original request, while preserving its background, material scope, required actions, and execution constraints. L0 appends all three preferences to the original request; L1 exposes only the scheduled current preference; L2 appends only a generic reminder to follow the user's usual habits; and L3 contains no preference hint. For every base family, L1 exposes P_1, P_2, and P_3 on Days 1, 2, and 3, respectively. The 40 Day 4 instances distribute their L1 exposure approximately evenly across P_1–P_3 (14/13/13). Irrespective of disclosure level, all three preferences are evaluated.

Evaluator construction and oracle calibration. Each task requirement is decomposed into weighted checks assigned to completion or to one of P_1–P_3. Deterministic checks inspect tool

results, access logs, final service state, and structured artifact values; semantic checks assess requirements such as grounded explanations or substantive coverage. The evaluator computes each axis from its checks and then applies equation 1.

We implement an executable reference policy for every instance using only the public L0 request and the declared tools. Each reference runs against a reset copy of the same mock environment as the evaluated agent. A reference is admitted only after all rule and semantic checks satisfy $C = E = P_1 = P_2 = P_3 = 1$. All 100 references satisfy this criterion. Their observed assistant-step counts define T_{ref}, and an input fingerprint binds each calibration to its prompts, data, sources, tools, and checks. Any change to these inputs invalidates the stored calibration. One assistant generation attempt, including the final response, counts as one step; multiple independent tool calls emitted in the same assistant message remain one step.

C Experimental Details

Baselines. All baselines follow the same three-day schedule.

- **Base Model**: The original model without external memory or parameter updates.
- **ReasoningBank** (Ouyang et al., 2026): Retrieves strategies from successful and failed trajectories.
- **MemEvolve** (Zhang et al., 2026a): Updates memory content and its management code. We use one candidate per day, with 16 base families for memory construction and 4 for validation using public-task success.
- **OpenClaw-RL** (Wang et al., 2026b): We use Binary RL with four trajectories per task and four steps per day. Step rewards use majority voting over three judgments from tool feedback.

We use GPT-5.6-Luna for all auxiliary-model calls, including memory operations for ReasoningBank and MemEvolve, process-reward inference for OpenClaw-RL, and benchmark judging for all three methods.

Evaluation. During learning, we evaluate STT before SFT, after SFT, and after RL on current-day base tasks and Day 4 tasks from all 40 families. Memory baselines are evaluated before each daily update. The terminal evaluation after Day 3 covers every available instance across Days 1–4 at L1–L3; generalization families contribute only their Day 4 instances. Each rollout starts from the same initial task state, and the environment is reset between independent samples. Benchmark scores, grader feedback, and privileged final state are never provided to the training-time trajectory scorer or repairer.

We first average the four independent rollouts for each task, then average tasks uniformly within each split and prompt level. The aggregate Day 4 score weights the base and generalization split means equally. Let T and T_{ref} be the actual and reference assistant-step counts. We use $E = \min(1, T_{\text{ref}}/T)$ when $T > 0$, and set $E = 0$ when $T = 0$.

Implementation Details. For each current-day L1 task, we collect four trajectories from the input policy at a temperature of 0.7 and select the first two by sample index, without using their benchmark scores. Each selected trajectory conditions one independently synthesized demonstration. The auxiliary model produces assistant actions, but every tool call is executed after resetting the task environment, so tool observations and final states cannot be fabricated. A candidate is revised until its public-task review and deterministic task and preference checks pass, or until the 16-round repair budget is exhausted. We exclude candidates that violate tool schemas, the 32-step limit, the per-response token limit, or the total context limit.

SFT uses all accepted demonstrations in one full batch per epoch and applies loss only to assistant tokens. Its learning rate follows cosine decay from 10^{-5} to 10^{-6}. Each RL update contains 20 task groups. For each task, the frozen current policy samples two original trajectories. The repairer iteratively replaces the first defective step and lets the frozen policy regenerate the suffix; the final counterpart of each original gives two original–repair pairs ($G = 4$). Incomplete repair pairs are dropped. The auxiliary scorer assigns an independent binary public-task outcome to every retained trajectory, and advantages use the sample mean and standard deviation within the task group. Zero-variance groups are masked; the optimizer step is skipped only when no group has nonzero advantage. The loss averages assistant tokens within each complete trajectory and then averages scored trajectories within the group, including auxiliary-model replacement steps.

We use Adam with $\beta_1 = 0.9$, $\beta_2 = 0.98$, weight decay 0.1, and gradient clipping at 1.0. RL uses a constant learning rate of 10^{-6}, lower and upper clipping thresholds of 0.2 and 0.28, and no KL or entropy loss. RL sampling uses temperature 1.0, top-$p = 1$, and no top-k truncation. Each trajectory permits at most 32 assistant steps; each step has at most 4,096 output tokens, within a 65,536-token context. All methods use these execution limits, and all reported runs use eight NVIDIA H800 GPUs.