AGENTBREW: LIFELONG KNOWLEDGE BREWING FROM STRONG TEACHERS TO WEAK LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deploying LLM agents typically requires a compact test-time student, even if a stronger teacher is available during training. We study knowledge brewing: distilling a teacher's interactive experience into a persistent external memory for the student. Crucially, this requires no weight updates, expert demonstrations, ground-truth labels, or test-time teacher access. This setting poses two challenges: environments provide only sparse, binary feedback, and teacher-authored notes must be inherently tailored to be concretely executable by a substantially weaker student. To address these hurdles, we propose **AgentBrew**, comprising two coupled components. First, a failure-triggered teacher–Ralph Loop mitigates sparse feedback by transforming student failures into environment-validated notes. Second, student-aware synthesis calibrates teacher knowledge to the weak executor's operational granularity, yielding model-specific, actionable guidance. Extensive evaluations and comprehensive ablations across coding, math, and tool-use tasks demonstrate that this asymmetric, training-free brewing paradigm produces highly capable yet deployable LLM agents. We have made our AgentBrew available anonymously at: https://anonymous.4open.science/r/AgentBrew-0200/.

1 INTRODUCTION

Large language model (LLM) agents can now solve interactive tasks in coding, mathematics, and tool-use environments (Gou et al., 2024; Trivedi et al., 2024; Jiang & Huang, 2026; Yang et al., 2024b). In production, however, the agent that serves requests is often a small or API-only model: fine-tuning may be unavailable, inference cost must remain low, and model weights are treated as fixed after deployment. A natural alternative is teacher–student agent brewing[1]: a stronger teacher agent $\mathcal{T}$ (a large LLM) distills interaction experience during a training period, whereas a compact student agent $\mathcal{S}$ (a small LLM) operates alone at test time with only retrieved external knowledge. The central question is how to convert brewing-time interaction into durable notes that improve $\mathcal{S}$'s effective capability without updating its parameters.

Two challenges make this setting substantially harder than self-evolution or standard distillation in isolation. **Challenge I: sparse environment feedback.** Real benchmarks and deployed APIs typically expose only a terminal binary reward $R \in \{0, 1\}$—pass or fail—together with the agent's own rollout, but no ground-truth solution, expert trajectory, or step-level label. When $\mathcal{S}$ fails, the only additional signal is the failure trajectory $\tau_{\mathcal{S}}(x)$; when it succeeds, no teacher intervention is required.

Challenge II: teacher–student capability gap (*i.e.*, knowledge is model-specific). In the deployment scenario above, $\mathcal{T}$ is intentionally much stronger than $\mathcal{S}$, yet the notes stored in memory must be written by $\mathcal{T}$ and executed by $\mathcal{S}$. Knowledge that a strong model can readily follow often fails in practice for a weak one: reasoning steps that $\mathcal{T}$ treats as obvious are omitted, vocabulary and concepts exceed what $\mathcal{S}$ can reliably apply, and corrective rules are too coarse to be turned into concrete actions. Figure 1 makes this dependence explicit: with Qwen3-14B fixed at test time, *S teacher + W student* brewing outperforms *S teacher + S student* brewing, showing that transferable notes depend on the student model used during brewing—not merely on teacher quality. Bridging this gap is therefore a first-order design problem.

[1]We use *brewing* to denote the offline process of distilling a teacher's knowledge into a persistent external memory that is validated on the target student. By *notes* we mean structured, retrievable corrective rules—conceptually similar to hints or procedural knowledge, but specifically validated to be executable by the student.

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

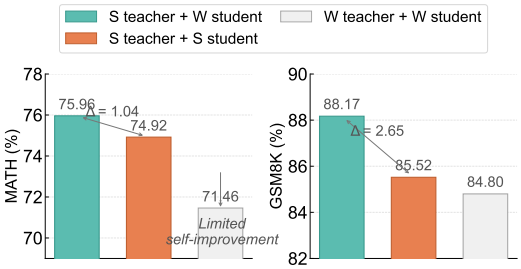


Figure 1: Accuracy on MATH and GSM8K under different brew-time teacher–student LLM pairings (test student: Qwen3-14B). Strong (S)/weak (W) denote DeepSeek-Chat-v3.1 and Qwen3-14B.

Existing work addresses only parts of the problem. Self-evolving agents (Shinn et al., 2023; Zhao et al., 2024) can improve from environment feedback without weight updates, but they usually (i) evolve the same model that both acts and learns, (ii) continue adapting at test time, and (iii) are not built for asymmetric teacher-student "brewing", where notes must be validated for a fixed weak executor. Agent distillation (Liu et al., 2025; Qiu et al., 2025a) can compress teacher behavior into trajectories or reusable artifacts, but it typically depends on labeled pairs or successful teacher rollouts, and it transfers knowledge at a teacher-level abstraction rather than as student-executable notes certified to work on the weak executor. The detailed discussion of related work is in App. A.1. Accordingly, we organize our work around three questions:

Q1 (Learning signal). When Env exposes only a binary reward R and no expert data, how can a weak agent extract useful knowledge from its failures, and how can that knowledge be verified before it is stored?

Q2 (Capability gap). How can a strong teacher author knowledge that a substantially weaker student can execute—not merely parse—when both its generation and its validation are biased by this gap?

Q3 (Lifelong accumulation). How can validated, student-calibrated knowledge accumulate across tasks in external memory—without updating the weights of $\mathcal{S}$—while keeping test-time inference lightweight and stable?

We propose a brew–serve framework (Figure 2) that maps Q1–Q3 to three coupled components: a failure-triggered **teacher–Ralph Loop**, **student-aware note synthesis**, and a **lifelong memory** $\mathcal{M}$. We call it **AgentBrew**: *brewing* is the Ralph Loop (Huntley, 2025) that iteratively distills certified notes from failures, and student-aware synthesis tailors each note to its executor, much as a brewer adapts the method to the ingredients. Brewing on $\mathcal{D}_{\text{train}}$ proceeds as follows. Student $\mathcal{S}$ rolls out each task in Env; when $R{=}0$, teacher $\mathcal{T}$ reads $(x, \tau_{\mathcal{S}}(x))$—the only supervision beyond binary pass/fail—and drafts a structured candidate note. The note is authored with student-aware prompting and an action-oriented schema, so that $\mathcal{T}$ targets $\mathcal{S}$'s reading level and operational granularity rather than $\mathcal{T}$'s. The Ralph Loop then stages the note and reruns $\mathcal{S}$ on the same task: a recovery ($R{=}1$) certifies the note from environment feedback alone, while guidance that remains plausible to $\mathcal{T}$ yet unusable by the weak executor is rejected. A curator deduplicates the certified notes and appends them to $\mathcal{M}$, which grows across the training stream without updating the parameters of $\mathcal{S}$.

Serving on $\mathcal{D}_{\text{test}}$ reads from the frozen memory $\mathcal{M}^{\star}$: $\mathcal{T}$ is inactive, and $\mathcal{S}$ retrieves skill-scoped top-k notes and completes each task in a single rollout, with no iterative reflection or online memory writes. AgentBrew is guided by a simple principle: **knowledge is not universal—it must fit the agent that executes it**. Our main contributions are:

- **Brew–serve under sparse feedback.** We develop a lifelong knowledge-building method for a fixed weak agent that uses only binary environment feedback, without expert trajectories, test-time labels, or any weight updates.
- **Student-calibrated transfer.** We introduce student-aware note synthesis, which proactively authors teacher knowledge at the operational granularity of $\mathcal{S}$, together with Ralph Loop recovery, a reactive filter that certifies notes by execution on the weak agent rather than by teacher plausibility alone.
- **Comprehensive evaluation.** We benchmark AgentBrew on three representative agent tasks (*i.e.*, coding, mathematics, and tool-use) under a shared brew–serve protocol, with ablations that assess the effect of each component.

2 Methodology

2.1 Problem Formulation

Setup. We consider interactive tasks x (*e.g.*, mathematics, coding, or tool-use). A fixed student agent $\mathcal{S}$, instantiated with a small LLM, interacts with an environment Env for up to H steps, yielding a trajectory $\tau_{\mathcal{S}}(x)$ and a binary terminal reward $R(x, \tau) \in \{0, 1\}$. The environment provides no ground-truth solutions, labels, or expert trajectories. During training on $\mathcal{D}_{\text{train}}$, a stronger teacher

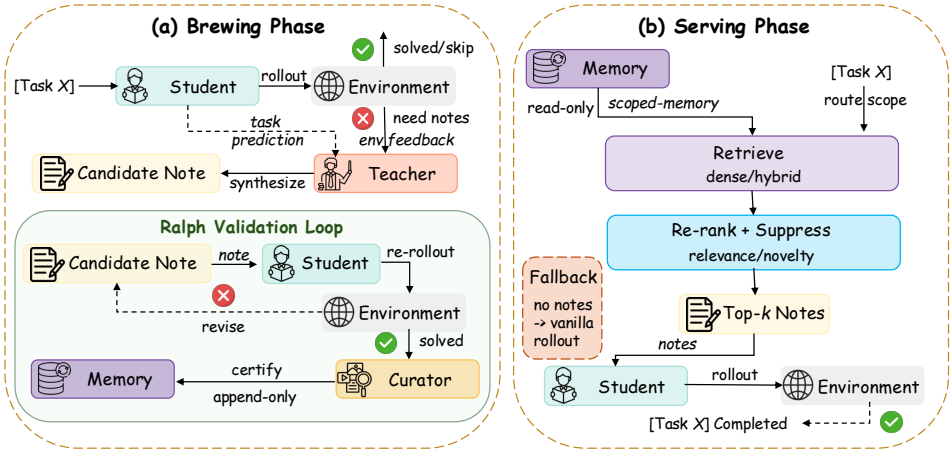


Figure 2: Overall framework of the proposed AgentBrew.

agent $\mathcal{T}$, instantiated with a large LLM, is available: upon any failure of $\mathcal{S}$, $\mathcal{T}$ analyzes the task and the failed trajectory and synthesizes structured notes, which are stored in an external memory $\mathcal{M}$. During testing on $\mathcal{D}_{\text{test}}$, $\mathcal{M}$ is frozen and $\mathcal{T}$ is entirely absent; $\mathcal{S}$ solves each task in a single pass using notes retrieved from $\mathcal{M}$. In this setting, learning is driven purely by environment feedback and failure trajectories, without offline demonstrations, test-time oracle labels, or updates to the parameters of $\mathcal{S}$.

Objective. Brew a persistent memory $\mathcal{M}^\star$ on $\mathcal{D}_{\text{train}}$ that maximizes held-out success:

$$\mathcal{M}^\star \in \arg\max_{\mathcal{M}} \mathbb{E}_{x\sim\mathcal{D}_{\text{test}}}\left[R\big(x,\, \tau_{\mathcal{S}}(x \mid \mathcal{M})\big)\right], \tag{1}$$

where $\tau_{\mathcal{S}}(x \mid \mathcal{M})$ is the trajectory rolled out by student agent $\mathcal{S}$ with the top-k notes from $\mathcal{M}$ prepended to its prompt. All lifelong gain is stored in $\mathcal{M}^\star$, not in the student agent's weights.

2.2 Framework Overview

AgentBrew operationalizes the objective in Sec. 2.1 via a *brew–serve* paradigm (Figure 2):

• **Brewing phase** *(on $\mathcal{D}_{\text{train}}$).* For each task x, the student agent $\mathcal{S}$ attempts x in Env. If R=1, the teacher is not invoked and the pipeline advances to the next task. If R=0, the teacher agent $\mathcal{T}$ reads $(x, \tau_{\mathcal{S}}(x))$ and, via student-aware prompting, proposes a structured candidate note n. The Ralph Loop (Huntley, 2025) injects n into $\mathcal{S}$ and re-rolls out on the *same* task x: a successful re-rollout (R=1) validates n, whereas persistent failure triggers revision or discard. Notes that pass the Ralph Loop are merged by a curator agent—performing deduplication and quality gating only, as correctness is already certified by the environment—and appended to $\mathcal{M}$. This cycle repeats over the training stream, progressively brewing $\mathcal{M}^\star$.

• **Serving phase** *(on $\mathcal{D}_{\text{test}}$).* At test time, memory is read-only and the teacher agent $\mathcal{T}$ is inactive. Given a new task x, $\mathcal{S}$ retrieves the top-k notes from frozen memory $\mathcal{M}^\star$ (via skill-scoped routing and reranking), prepends them to its prompt, and completes x in a *single* environment rollout-without an iterative reflection loop or additional validation calls. If retrieval returns no notes, $\mathcal{S}$ falls back to its vanilla policy.

2.3 From Binary Feedback to Validated Knowledge

The brewing phase must extract durable knowledge from failures when Env exposes only a terminal reward $R \in \{0, 1\}$: no step-level labels, no expert trajectories, and no oracle explanations. We address **Q1** by treating each failure as a supervision event. Here we detail the failure-triggered activation pipeline, the note representation written by $\mathcal{T}$, and curation before durable writes.

Failure-Triggered Teacher Activation. During brewing, each task $x \in \mathcal{D}_{\text{train}}$ begins with a rollout by student agent $\mathcal{S}$, producing $\tau_{\mathcal{S}}(x)$ and reward $R \in \{0, 1\}$. Binary R is the sole external supervision signal from Env: no step-level labels, no expert trajectories, and no oracle explanations are provided. If $R = 1$, the pipeline advances to the next task without invoking $\mathcal{T}$. If $R = 0$, we trigger $\mathcal{T}$ on the

tuple $(x,\ \tau_{\mathcal{S}}(x),\ \text{FAIL})$. Unlike Supervised Fine-Tuning (SFT), $\mathcal{T}$ never conditions on a reference solution; it diagnoses the failure mode from the student trajectory and task text alone, then proposes a corrective note. This design makes learning possible in settings where only pass/fail feedback is available. Algo. 1 summarizes the control flow.

Algorithm 1: Failure-triggered teacher activation on $\mathcal{D}_{\text{train}}$.

Input: $\mathcal{D}_{\text{train}}$, student $\mathcal{S}$, teacher $\mathcal{T}$, memory $\mathcal{M}$, environment Env

for *each* $x \in \mathcal{D}_{\text{train}}$ **do**
  $\mathcal{N} \leftarrow \textsc{Retrieve}(\mathcal{M}, x)$ `// retrieved notes for` x `(may be` $\emptyset$`)`
  $\tau_{\mathcal{S}}(x) \leftarrow \textsc{Rollout}(\mathcal{S}, x, \mathcal{N})$
  $R \leftarrow \textsc{Eval}(\text{Env}, x, \tau_{\mathcal{S}}(x))$ `//` $R \in \{0,1\}$`; sole external supervision from Env`
  if $R = 1$ **then**
    continue `// success:` $\mathcal{T}$ `not invoked`
  end
  else
    $n \leftarrow \textsc{DraftNote}(\mathcal{T}, x, \tau_{\mathcal{S}}(x), \text{FAIL})$ `// no reference solution`
    $\textsc{RalphLoop}(x, n, \mathcal{S}, \mathcal{T}, \mathcal{C}, \mathcal{M}, \text{Env})$ `// refinement and curation`
  end
end

Structured Note Generation. Given a failure, $\mathcal{T}$ outputs a candidate note n as structured JSON rather than a worked example. Each note is an abstract, reusable rule intended for retrieval on future tasks with similar structure:

- `note_key`: unique identifier (reused when updating an existing rule);
- `trigger_pattern`: task features that should activate the note;
- `corrective_rule`: an executable fix directive for $\mathcal{S}$;
- `minimal_steps`: a short ordered checklist lowering interpretation cost for a weak LLM;
- `tags`: domain or skill labels for routing;
- `quality_score`: initial confidence, later updated from validation statistics.

Notes deliberately avoid storing full solutions to x; they compress failures into transferable guardrails. Granularity and wording are calibrated to the student agent's capability.

Note Curator. When the Ralph Loop returns $validated = \text{TRUE}$, Algo. 1 passes the environment-certified note to curator agent $\mathcal{C}$ before any durable write. $\mathcal{C}$ does *not* re-judge factual correctness—recovery in Env already certifies utility to $\mathcal{S}$ on the brewing task x. Its role is strictly operational on the *memory store*:

(i) ***Deduplication:*** compare n against existing entries in $\mathcal{M}$ via lexical and semantic similarity; merge near-duplicates by updating an existing `note_key` instead of appending redundant rules.

(ii) ***Quality tracking:*** adjust `quality_score` from historical validation outcomes (acceptances, recoveries, and downstream usage).

(iii) ***Write gating:*** persist n to $\mathcal{M}$ only if it passes redundancy and quality thresholds; otherwise discard despite a successful Ralph Loop round.

Accepted notes are appended under an append-only policy, extending $\mathcal{M}$ without overwriting prior knowledge or modifying the student LLM.

2.4 Bridging the Capability Gap

Teacher agent $\mathcal{T}$ is substantially stronger than student agent $\mathcal{S}$, leading to the capability gap (Li et al., 2025) between two agents. We address **Q2** with a *two-stage* design. **First, reactive validation**: the **Ralph Loop** (Huntley, 2025) re-rolls $\mathcal{S}$ on the failed task with a candidate note staged in memory and accepts n only when Env returns R=1, using terminal reward—not teacher or student self-judgment—as the certification signal. **Second, proactive synthesis**: student-aware prompting and note schema calibrate what $\mathcal{T}$ writes before validation, so proposals match $\mathcal{S}$'s reading level and operational granularity. Ralph provides the environment-grounded filter; proactive synthesis improves validation yield and transferability.

• **Why same-level generation fails.** Prior work on knowledge transfer, such as ACE (Zhang et al., 2025), assumes that the knowledge producer and consumer have comparable reasoning ability. When

$\mathcal{T}$ is substantially stronger than $\mathcal{S}$, this assumption breaks down (Chen et al., 2025; Mirzadeh et al., 2020): notes often omit intermediate reasoning, use mismatched vocabulary, or specify steps at a granularity too coarse for $\mathcal{S}$ to follow. Such notes may appear plausible to $\mathcal{T}$ yet leave $\mathcal{S}$ unable to recover.

Reactive Validation: Ralph Loop. Teacher-generated notes can be plausible yet useless or harmful to $\mathcal{S}$. Self-judgment (Shinn et al., 2023) by $\mathcal{T}$ or $\mathcal{S}$ is insufficient because capability gaps bias both generation and critique. The Ralph Loop is the environment-validation stage of brewing: it asks whether a candidate note actually helps $\mathcal{S}$ recover on the failed task, using only terminal rewards from Env.

Provisional staging. After $\mathcal{T}$ proposes n, ADDPROVISIONAL$(\mathcal{M}, n)$ places n in a *provisional* slot—visible to RETRIEVE$(\mathcal{M}, x)$ during re-attempts on x, but not yet part of durable long-term memory. Provisional entries simulate "what if this note were already in $\mathcal{M}$?" for the validation rollouts; ROLLBACKPROVISIONAL removes the slot if validation fails, and COMMIT promotes a curated note only after $\mathcal{C}$ approves persistence.

Environment validation. With n staged provisionally, we re-run $\mathcal{S}$ on the *same* task x, retrieving $\mathcal{N}$ that includes n along with any notes already committed for x. Let $\tau_{\mathcal{S}}^{(r)}(x \mid n)$ denote the r-th re-attempt trajectory and $R^{(r)}$ its terminal reward. The Ralph Loop accepts n when recovery occurs:

$$R^{(r)} = 1 \quad \text{for some re-attempt } r \in \{1, \ldots, R_{\max}\}, \tag{2}$$

where $R_{\max}$ caps extra environment calls. If no attempt succeeds, ROLLBACKPROVISIONAL$(\mathcal{M})$ discards the staging slot; optionally $\mathcal{T}$ revises n from the latest failed trajectory $\tau^{(r)}$ before the next r. On success, the loop returns $validated = \text{TRUE}$ and leaves n provisionally staged for $\mathcal{C}$; Algo. 1 then invokes curation.

Algo. 2 details the validation procedure. It uses only $R^{(r)}$ from Env—never ground-truth labels on $\mathcal{D}_{\text{test}}$ and never oracle answers during serving. At test time, no Ralph Loop calls are made; validated knowledge is consumed from the frozen $\mathcal{M}^{\star}$ in a single rollout.

Algorithm 2: Ralph Loop: environment validation of a candidate note on task x.

Input: task x, candidate note n, agents $\mathcal{S}, \mathcal{T}$, memory $\mathcal{M}$, environment Env, budget $R_{\max}$
Output: $validated \in \{\text{TRUE}, \text{FALSE}\}$

ADDPROVISIONAL$(\mathcal{M}, n)$ // stage n for RETRIEVE only; not durable yet
$validated \leftarrow$ FALSE
for $r = 1$ **to** $R_{\max}$ **do**
  $\mathcal{N} \leftarrow$ RETRIEVE$(\mathcal{M}, x)$
  $\tau^{(r)} \leftarrow$ ROLLOUT$(\mathcal{S}, x, \mathcal{N})$
  $R^{(r)} \leftarrow$ EVAL$(\text{Env}, x, \tau^{(r)})$
  if $R^{(r)} = 1$ **then**
    $validated \leftarrow$ TRUE
    break // satisfies Eq. 2
  end
  if $r < R_{\max}$ **then**
    $n \leftarrow$ DRAFTNOTE$(\mathcal{T}, x, \tau^{(r)}, \text{FAIL})$ // revise n from failed re-attempt
    UPDATEPROVISIONAL$(\mathcal{M}, n)$ // replace staged copy only
  end
end
if $\neg validated$ **then**
  ROLLBACKPROVISIONAL$(\mathcal{M})$ // no recovery; discard staged n
end
return $validated$ // on success, n remains provisional for $\mathcal{C}$

Proactive Synthesis: Student-Aware Teacher Prompting. Ralph validation alone is a costly filter: when $\mathcal{T}$ drafts ACE-style bullets that omit steps $\mathcal{S}$ cannot infer, most proposals fail within $R_{\max}$. We therefore instruct $\mathcal{T}$ to treat $\mathcal{S}$ as the sole intended reader: notes must read like a checklist for a junior agent, not an expert post-mortem. Given $(x, \tau_{\mathcal{S}}(x))$ from a failed rollout, $\mathcal{T}$ follows a three-step synthesis protocol before emitting JSON:

(i) ***Failure-mode identification:*** Parse $\tau_{\mathcal{S}}(x)$ and the environment report to label the dominant error type (*e.g.*, wrong API call order, sign error, misread constraint).

(ii) ***Capability-gap localization:*** Attribute the error to a *specific* missing skill or knowledge fragment at $\mathcal{S}$'s level—not a vague "logical mistake"—distinguishing representation errors from execution errors where possible.

(iii) ***Executable rule extraction:*** Emit a corrective procedure in imperative form ("first do A, then B, watch for C"), banning appeals to tacit understanding ("deeply analyze X") that $\mathcal{S}$ cannot operationalize.

The prompt also states $\mathcal{S}$'s model class and known weaknesses (*e.g.*, limited multi-step arithmetic, brittle tool syntax), biasing $\mathcal{T}$ toward shorter sentences, explicit ordering, and surface-level triggers grounded in $\tau_{\mathcal{S}}(x)$.

2.5 Lifelong Memory

AgentBrew addresses **Q3** by treating external memory $\mathcal{M}$ as the sole durable state: validated notes accumulate across $\mathcal{D}_{\text{train}}$, and student agent $\mathcal{S}$ reads from a frozen $\mathcal{M}^{\star}$ at test time. No gradient updates or adapter weights are applied; effective capability grows because $\mathcal{M}$ grows. We structure $\mathcal{M}$ to limit cross-domain interference, append new knowledge without erasing old entries, and retrieve notes with a serving pipeline tuned for single-pass inference.

Skill-Scoped Memory Architecture. A monolithic note pool induces *cross-scope interference*: notes from one skill are retrieved on unrelated tasks. We therefore partition $\mathcal{M}$ into disjoint skill scopes $\{\mathcal{M}_s\}_{s\in\mathcal{S}}$, each indexing notes for a single capability or environment slice. Every note stores `tags` and an optional scope identifier; during curation, the teacher writes it to the corresponding $\mathcal{M}_s$. At test time, retrieval first selects a scope subset $\mathcal{M}_{\text{scope}} \subseteq \mathcal{M}$ and ranks notes only within that subset-never over the full pool.

Scope routing is orthogonal to note ranking. When x does not clearly specify a scope, we choose $\mathcal{M}_{\text{scope}}$ using three signals, in order: **(i)** explicit scope or tool names in x; **(ii)** lexical or semantic similarity between x and a scope-description catalog maintained alongside $\mathcal{M}$; **(iii)** actions observed in the student rollout $\tau_{\mathcal{S}}(x)$ when (i)–(ii) are inconclusive.

Continual Accumulation and Stability. During brewing, accepted notes are merged with **append-only** semantics: curator deduplication may *update* an existing `note_key`, but validated rules are never deleted to make room for new ones, avoiding catastrophic forgetting of earlier skills. Each note record tracks usage statistics and `quality_score` so frequently harmful or redundant entries can be down-weighted at retrieval time without erasing history.

Train and test streams are strictly separated. While processing $\mathcal{D}_{\text{train}}$, $\mathcal{M}$ is writable. On $\mathcal{D}_{\text{test}}$, $\mathcal{M}^{\star}$ is **frozen read-only**: no teacher calls, no provisional writes, and no statistic updates that leak test-task content back into the store. This guarantees that performance on held-out tasks measures generalization from brewed knowledge, not online adaptation to test labels or test-task feedback.

Test-Time Knowledge-Augmented Inference. Serving reuses the objective in Eq. 1 with a fixed retrieval-and-rollout procedure (Appendix A.2). Given task x, we encode a query $q(x)$ from the instruction, select the skill scope, and retrieve candidate notes from $\mathcal{M}^{\star}_{\text{scope}}$ using dense or hybrid lexical–semantic scoring. Candidates are reranked by

$$\text{score}(n, x) = \text{sim}(q(x), n) \cdot \rho(n) \cdot \phi(n), \tag{3}$$

where $\text{sim}(\cdot)$ is retrieval similarity, $\rho(n)$ is the stored `quality_score` (and success-rate prior), and $\phi(n)$ is a frequency penalty that down-weights notes over-selected in past tasks to promote diversity. Rather than hard-dropping low-scoring notes, **soft suppression** applies $\phi(n)$ and a relative-score floor: notes below a fraction of the top score are dampened but may still appear when the pool is sparse, reducing brittle all-or-nothing retrieval. The top-k notes are formatted and prepended to $\mathcal{S}$'s prompt; $\mathcal{S}$ then performs one ROLLOUT in Env. If retrieval returns an empty set, $\mathcal{S}$ falls back to its vanilla policy with no memory augmentation.

3 Evaluation

3.1 Experimental Setup

Tasks and Benchmarks. We evaluate AgentBrew across three representative agent scenarios—**coding**, **math**, and **tool-use**—each probing a distinct capability profile: synthesizing executable

programs from natural-language specifications, performing multi-step symbolic reasoning, and completing long-horizon tasks through interactive API calls. More details are in App. A.3.

• For the **coding** tasks, we evaluate AgentBrew on *MBPP* (Austin et al., 2021), a benchmark of roughly 1,000 entry-level Python programming tasks, and *MBPP+* (Liu et al., 2023), an extension that applies stricter unit testing to a curated subset to reduce spurious passes.

• For the **math** tasks, we evaluate AgentBrew on *MATH* (Hendrycks et al., 2021), comprising challenging competition-level problems, and *GSM8K* (Cobbe et al., 2021), which tests multi-step grade-school arithmetic reasoning. We score final-answer accuracy using the Qwen2.5-Math (Yang et al., 2024a) evaluation framework.

• For the **tool-use** tasks, we evaluate AgentBrew on *AppWorld* (Trivedi et al., 2024), an interactive benchmark where agents write and execute Python code to complete realistic, multi-app daily tasks in a simulated environment with multiple APIs.

Models, Baselines and Metrics. • **LLM Base Models.** In our experiments, we use `deepseek-chat-v3.1` (Liu et al., 2024) by default for the teacher agent and `qwen3-14b` (Yang et al., 2025) by default for the student agent.

• **Baseline Methods.** The main baseline methods we compare against include: **Simple RAG**, **ExpeL** (Zhao et al., 2024), **Reflexion** (Shinn et al., 2023), **ACE** (Zhang et al., 2025), **ReAct** (Yao et al., 2023), and **LoRA** (Hu et al., 2022). Among these, **Simple RAG** refers to saving the successful trajectories of the teacher agent during training, and directly retrieving these trajectories during testing to help the student agent complete the task.

• **Evaluation Metrics.** We use accuracy for math tasks, the strict EvalPlus (Liu et al., 2023) benchmark for coding tasks, and Task Goal Completion (TGC) for tool-use tasks (*i.e.*, AppWorld) as our evaluation metrics.

3.2 Main Performance Evaluation

Table 1: Main Results on Three Agent Scenarios. (* indicates AgentBrew significantly outperforms the strongest baseline in Group I and Group III at $p<0.05$ under a two-proportion z-test.)

Method	Teacher	Test-time	Math		Coding		Tool-Use	
			MATH (%)	GSM8K (%)	MBPP (%)	MBPP+ (%)	Normal (%)	Challenge (%)
Group I: *Student-Only, Single Rollout*								
ReAct	✗	1 rollout	71.66	86.12	51.02	68.83	22.00	11.30
SFT(LoRA)[2]	✗	1 rollout	61.50	87.11	–	–	–	–
Group II: *Student-Only, Multi-Rollout + Test-Time Feedback*								
Reflexion (k=2)	✗	2 rollouts + fb	77.12	91.51	54.22	70.78	29.76	14.63
Reflexion (k=3)	✗	3 rollouts + fb	80.06	93.10	60.54	70.13	36.31	17.75
Group III: *Teacher-Augmented Brewing, Single Rollout*								
Simple RAG	✓	1 rollout	74.32	82.56	60.54	71.43	29.17	13.19
ExpeL	✓	1 rollout	70.60	88.10	61.22	68.18	28.57	10.80
ACE	✓	1 rollout	73.82	87.87	58.61	64.29	26.79	10.55
AgentBrew	✓	1 rollout	**75.96***	**88.17***	**61.22**	**71.43***	**31.55***	**15.35***

Table 1 reports main results on all six benchmarks. Unless noted otherwise, every method uses Qwen3-14B as the test-time *generator*. Critically, all brewing is performed exclusively on $\mathcal{D}_{\text{train}}$, while evaluation uses held-out $\mathcal{D}_{\text{test}}$ tasks. The consistent gains over the vanilla student across all three task families on unseen test problems provide direct evidence that **brewed notes capture transferable heuristics rather than task-specific patches**.

(i) Strong-teacher brewing *vs.* student-only reflection. ReAct and Reflexion (Groups I–II) rely entirely on the weak student at test time: ReAct acts in a single pass, while Reflexion adds multiple reflection rounds that re-attempt each task after observing explicit binary feedback from the environment. The Reflexion setup with $k>1$ is therefore a stronger inference protocol than our single-rollout serving (Group III), as it consumes additional test-time trials and outcome signals that are unavailable to all other methods. Even so, AgentBrew clearly outperforms ReAct across all three scenarios, and despite using only one test-time attempt, matches or exceeds Reflexion (k=2) on all four coding and tool-use benchmarks. Reflexion's advantage is concentrated on math, where iterative self-correction from explicit pass/fail signals is most beneficial for multi-step arithmetic reasoning. These trends

indicate that routing binary failure feedback through a strong teacher and validating notes on the target student yields more useful knowledge than unaided self-reflection, particularly for complex code generation and API interaction where weak-model reflection is insufficient.

(ii) Student-calibrated transfer *vs.* teacher-agnostic memory. Simple RAG, ExpeL, and ACE also use a strong teacher to build external memory (Group III), but none explicitly calibrates that memory to the weak student or validates it on the executor that must use it at test time. AgentBrew consistently outperforms ACE across mathematics, coding, and tool-use, and ranks best or tied for best against Simple RAG and ExpeL on most benchmarks—with particularly clear gains on harder coding and tool-use splits such as MBPP+ and AppWorld `test_challenge`. These results indicate that bridging the teacher–student capability gap through student-aware authoring and environment-side validation is key to turning teacher knowledge into guidance that a fixed weak agent can execute.

3.3 Ablation Study

To isolate the contribution of individual components, we ablate AgentBrew on the math scenario (MATH and GSM8K) while holding the test-time student fixed (*i.e.*, Qwen3-14B). Table 2 compares the full system against four variants, each modifying one part of AgentBrew.

Variant settings. • *w/o Ralph Loop*: The teacher drafts notes upon student failure without performing Ralph recovery checks. • *w/o Memory*: The external memory is disabled; no notes are stored or retrieved at test time. • *w/o Test-Time Suppression*: Serving relies solely on brewing-time quality scores, omitting soft suppression during retrieval. • *w/ Strong Student Knowledge*: A strong LLM (DeepSeek-Chat-V3.1) replaces the weak student during the brewing phase, while test-time inference continues to use Qwen3-14B.

Table 2: Ablation Study on MATH Scenario.

Variant	MATH (%)	GSM8K (%)
w/o Ralph Loop	75.58	84.61
w/o Memory	71.66	86.12
w/o Test-Time Suppression	74.98	84.84
w Strong Student Knowledge	74.92	85.52
AgentBrew	75.96	88.17

Findings. • *w/o Ralph Loop* (MATH: 75.58, GSM8K: 84.61) lags behind the full system, particularly on GSM8K, confirming that recovery checks successfully filter out notes that are unhelpful to the weak student. • *w/o Memory* yields the lowest MATH score (71.66) while maintaining GSM8K performance (86.12), highlighting that external memory is vital for complex reasoning tasks even if the student excels at basic arithmetic. • *w/o Test-Time Suppression* (MATH: 74.98, GSM8K: 84.84) underperforms the full system, showing that soft suppression successfully down-weights notes that score highly during training but lack retrieval-time relevance. • *w/ Strong Student Knowledge* (MATH: 74.92, GSM8K: 85.52) also trails the full system, demonstrating that notes tailored to a strong surrogate student transfer ineffectively to Qwen3-14B during inference.

3.4 Deep Analysis of Ralph Loop

To assess the Ralph Loop beyond end-task accuracy, we compare Full Ralph with a *w/o Ralph* variant that proposes notes without recovery-based certification. The results in Table 3 show a modest gain in training success and a much larger change in how failures are converted into committed notes.

During training, the Ralph Loop directly improves task-solving performance, with Full Ralph achieving success rates of 92.2% on MATH and 91.3% on GSM8K (compared to 89.6% and 87.6% *w/o Ralph*). More importantly, Ralph fundamentally changes knowledge accumulation. Despite encountering *more* training failures (104 vs. 78 on MATH; 124 vs. 87 on GSM8K), the *w/o Ralph* variant produces *fewer* final notes (27 vs. 39 on MATH; 33 vs. 43 on GSM8K). The root cause is the note conversion rate: Full Ralph translates 61.5% and 75.9% of failures into committed notes, while *w/o Ralph* converts a mere 26.9% and 38.7%.

This efficiency gap arises from differing validation mechanisms. Without Ralph, notes are generated post-hoc without proof of utility, forcing the curator to act as the sole quality gate and reject 61–73% of them. In Full Ralph, the loop acts as an implicit pre-filter: provisional notes reach the Curator only

[2]SFT(LoRA) is evaluated only on math, where ground-truth answers are available; coding and tool-use tasks lack reference solutions for supervised training.

Table 3: Training-phase statistics for Full Ralph vs. *w/o Ralph* on MATH and GSM8K.

	MATH		GSM8K	
Metric	**Full Ralph**	***w/o* Ralph**	**Full Ralph**	***w/o* Ralph**
Training success rate	92.2%	89.6%	91.3%	87.6%
Failed tasks	78	104	87	124
Ralph committed tasks	28	0	44	0
Direct-write tasks	20	28	22	48
Total write tasks	48	28	66	48
Note write rate	61.5%	26.9%	75.9%	38.7%
Curator rejections	30	76	21	76
Curator rejection rate	38.5%	73.1%	24.1%	61.3%
Notes number	39	27	43	33

after demonstrably helping the student recover. This student-validated signal cuts the rejection rate to 24–39% and guarantees that every committed note is empirically effective. The result is a richer, reliably grounded knowledge base that drives Full Ralph's superior test-time performance.

3.5 Effectiveness of Knowledge Brewing on Different LLMs

To determine whether AgentBrew relies on specific model pairings, we evaluate various combinations of *teachers* (active only on $\mathcal{D}_{\text{train}}$) and *students* (active at brewing and serving) on AppWorld `test_normal`, reporting TGC. Holding the rest of the pipeline fixed, we compare three teachers (GLM-5.1 (Zeng et al., 2026), DeepSeek-Chat-v3.1, and *None*) against two student backbones (DeepSeek-Chat-v3.1 and Qwen3-14B). Figure 3 shows the results, yielding the following insights:

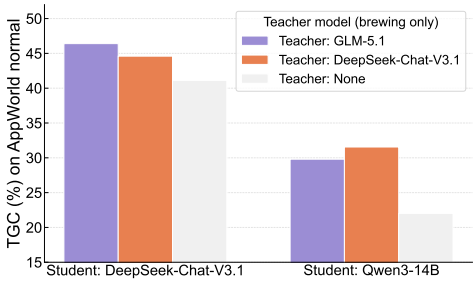


Figure 3: Performance on AppWorld `test_normal` under different teacher–student LLM pairings.

(i) A strong teacher aids diverse LLM pairings. Across both student backbones, invoking a strong teacher during brewing strictly outperforms the *None* baseline (where students learn only from their own failures). TGC increases from 41.1% to 46.4% for DeepSeek-Chat-v3.1, and from 22.0% to 31.55% for Qwen3-14B. These gains confirm that AgentBrew robustly generalizes across different vendor and capability combinations.

(ii) Teacher strength does not strictly dictate transfer success. While strong teachers consistently beat the baseline, optimal teacher–student pairings vary: DeepSeek-Chat-v3.1 pairs best with Qwen3-14B (31.55% vs. 29.8%), while GLM-5.1 pairs best with DeepSeek-Chat-v3.1 (46.4% vs. 44.6%). The smaller gap between teachers on Qwen3-14B likely stems from the student's limited execution capacity—notes that pass brewing may still fail at serving time, making differences in teacher quality harder to realize. This observation strongly motivates AgentBrew's student-side validation: rather than assuming teacher-optimal text is universally beneficial, notes must be explicitly validated against the target student's capabilities.

Moreover, we provide a qualitative analysis of brewed note generalization in App. A.4, study the scaling ability in App. A.6, and present a real-world deployment case study in App. A.5.

4 Conclusion

In this paper, we present AgentBrew, a training-free framework in which a strong teacher distills interaction experience into an external memory for a fixed, weaker student. We show that effective knowledge transfer hinges on validating notes against the student's actual capabilities rather than the teacher's abstractions. With consistent gains across coding, mathematics, and tool-use tasks, AgentBrew establishes asymmetric knowledge brewing as a practical path to deployable LLM agents.

AI USE STATEMENT

In this work, we used generative AI tools solely to edit the manuscript and improve its readability, including grammar, wording, and clarity of presentation. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, provide ingredients for proofs, assist in writing proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, translate the paper, clean or reformat datasets, support qualitative or thematic data analysis, or interpret results; these uses are not applicable to this work. We have reviewed all AI-assisted edits and verified that they do not alter the scientific claims, experimental results, or technical content of the paper. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Anthropic. Claude code. `https://docs.anthropic.com/en/docs/claude-code`, 2025.

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.

Justin Chih-Yao Chen, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. Magdi: Structured distillation of multi-agent interaction graphs improves reasoning in smaller language models. *arXiv preprint arXiv:2402.01620*, 2024.

Xinghao Chen, Zhijing Sun, Guo Wenjin, Miaoran Zhang, Yanjun Chen, Yirong Sun, Hui Su, Yijie Pan, Dietrich Klakow, Wenjie Li, et al. Unveiling the key factors for distilling chain-of-thought reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 15094–15119, 2025.

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.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: On path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 1, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations*, volume 2024, pp. 48362–48395, 2024.

Maryam Hashemzadeh, Elias Stengel-Eskin, Sarath Chandar, and Marc-Alexandre Cote. Sub-goal distillation: A method to improve small language agents. *arXiv preprint arXiv:2405.02749*, 2024.

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. *arXiv preprint arXiv:2103.03874*, 2021.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

Geoffrey Huntley. Ralph wiggum as a "software engineer". Blog post, 2025. URL `https://ghuntley.com/ralph/`. Defines the Ralph Loop: an iterative bash loop feeding a fixed prompt to a coding agent with externalized state (files, git, tests).

Humaid Ibrahim, Nikolai Rozanov, and Marek Rei. Fine-tuning with rag for improving llm learning of new skills. *arXiv preprint arXiv:2510.01375*, 2025.

Xun Jiang, Feng Li, Han Zhao, Jiahao Qiu, Jiaying Wang, Jun Shao, Shihao Xu, Shu Zhang, Weiling Chen, Xavier Tang, et al. Long term memory: The foundation of ai self-evolution. *arXiv preprint arXiv:2410.15665*, 2024.

Yangqin Jiang and Chao Huang. Openphone: Mobile agentic foundation models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 30362–30380, 2026.

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, 2026.

Yuetai Li, Xiang Yue, Zhangchen Xu, Fengqing Jiang, Luyao Niu, Bill Yuchen Lin, Bhaskar Ramasubramanian, and Radha Poovendran. Small models struggle to learn from strong reasoners. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 25366–25394, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

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.

Jun Liu, Zhenglun Kong, Peiyan Dong, Changdi Yang, Tianqi Li, Hao Tang, Geng Yuan, Wei Niu, Wenbin Zhang, Pu Zhao, et al. Structured agent distillation for large language model. *arXiv preprint arXiv:2505.13820*, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 5191–5198, 2020.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

Jiahao Qiu, Xinzhe Juan, Yimin Wang, Ling Yang, Xuan Qi, Tongcheng Zhang, Jiacheng Guo, Yifu Lu, Zixin Yao, Hongru Wang, et al. Agentdistill: Training-free agent distillation with generalizable mcp boxes. *arXiv preprint arXiv:2506.14728*, 2025a.

Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, et al. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution. *arXiv preprint arXiv:2505.20286*, 2025b.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. Distilling reasoning capabilities into smaller language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 7059–7073, 2023.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

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.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.03629.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

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. *arXiv preprint arXiv:2510.04618*, 2025.

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

Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, et al. Ama-bench: Evaluating long-horizon memory for agentic applications. *arXiv preprint arXiv:2602.22769*, 2026.

A Appendix

A.1 Related Work

A.1.1 Self-Improving Agents from Environment Feedback

LLM agents interleave language reasoning with environment actions (*e.g.*, ReAct (Yao et al., 2023) and Toolformer-style tool-use (Schick et al., 2023)). Beyond one-shot prompting, a self-evolving agent literature studies how agents improve from interaction rather than offline expert labels, often leveraging sparse environment feedback as the main supervision for accumulating reusable competence (Gao et al., 2025). Representative approaches differ in what they evolve and what feedback they require: verbal self-improvement systems treat language as mutable memory by writing critiques after failures (*e.g.*, Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023)), while experience-driven skill libraries extract reusable procedures from interaction to grow pools or code-based repositories (*e.g.*, ExpeL (Zhao et al., 2024) and Voyager (Wang et al., 2023)). More modular self-evolution and long-horizon memory frameworks push adaptation toward reusable protocols or persistent external stores (*e.g.*, Alita (Qiu et al., 2025b) and OMNE (Jiang et al., 2024)), and playbook-style context engineering closes execution-reflection-curation loops by revising structured bullets (*e.g.*, ACE (Zhang et al., 2025)). Overall, most self-evolving agents assume same-capability evolution and adapt online at test time, learning from successes or rich feedback, whereas AgentBrew instead performs offline "brewing" of environment-validated, student-aware notes under binary supervision only, and then serves them via a frozen memory at test time.

A.1.2 Knowledge Transfer and Lifelong Memory for LLM Agents

Knowledge distillation has been widely used to compress large language models and has recently been extended to agent settings by transferring behavior from a teacher to a student across different transfer channels (Hinton et al., 2015; Shridhar et al., 2023). At the trajectory level, structured span supervision and retrieval-augmented rollouts train students to imitate interleaved reasoning-action traces (Liu et al., 2025; Kang et al., 2026). In contrast, structure-level distillation compresses trajectories into interaction graphs or subgoal plans to preserve high-level task structure (Chen et al., 2024; Hashemzadeh et al., 2024). Training-free approaches further store reusable external artifacts: for example, AgentDistill extracts Model-Context-Protocols (MCPs) from successful teacher trajectories into an inference-time MCP-Box (Qiu et al., 2025a). RAG Distillation similarly distill teacher-to-student knowledge from failure-extracted hints, but internalize it into model weights via fine-tuning at the teacher's abstraction level, without calibrating knowledge to the student's capability or supporting incremental accumulation without re-training (Ibrahim et al., 2025). Related memory-based and lifelong-retrieval settings maintain external knowledge without parameter updates (Jiang et al., 2024; Packer et al., 2023; Zhao et al., 2026; Trivedi et al., 2024).Overall, our work addresses the limitations of protocol-like artifact transfer and teacher-dependent abstractions by generating student-validated corrective notes from student failures using only binary rewards.

A.2 Test-Time Serving Process.

Algo. 3 formalizes the test-time inference procedure introduced in Sec. 2.5 and Sec. 2.2. For each test task x, the instruction is first encoded into a retrieval query q that captures key textual signals such as task descriptions and tool names (line 2). The skill-scoped router RouteScope then selects a partition $\mathcal{M}_{\text{scope}} \subseteq \mathcal{M}^{\star}$ following the scoping policy in Sec. 2.5—explicit scope hints in x are matched first, then lexical similarity against category descriptions, and finally runtime cues when available—confining subsequent retrieval to a relevant skill subset and preventing cross-domain interference (line 3). Candidate notes $\mathcal{C}$ are drawn from $\mathcal{M}_{\text{scope}}$ via dense or hybrid similarity scoring and reranked via Eq. 3, which modulates each note's retrieval similarity by its historical `quality_score` $\rho(n)$ and a frequency penalty $\phi(n)$; soft suppression dampens notes below a relative-score floor without hard-dropping them, preserving recall when the candidate pool is sparse. The top-k notes $\mathcal{N}_k$ are then formatted and prepended to $\mathcal{S}$'s system prompt (lines 4–5). If $\mathcal{N}_k$ is non-empty, $\mathcal{S}$ performs a single Rollout with memory augmentation; if retrieval returns no notes, the system falls back to the student's vanilla policy with no augmentation (lines 6–8). Critically, no teacher agent, Ralph Loop validation, or provisional memory writes are invoked during serving—the frozen $\mathcal{M}^{\star}$ is consumed read-only, and every task is completed in exactly one rollout, matching

Algorithm 3: Test-time serving on $\mathcal{D}_{\text{test}}$ with frozen $\mathcal{M}^{\star}$.

Input: $\mathcal{D}_{\text{test}}$, frozen memory $\mathcal{M}^{\star}$, student agent $\mathcal{S}$, environment Env, top-k k, rerank params ρ, ϕ

1 **for** *each task* $x \in \mathcal{D}_{\text{test}}$ **do**
2 $q \leftarrow \textsc{EncodeQuery}(x)$
3 $\mathcal{M}_{\text{scope}} \leftarrow \textsc{RouteScope}(x, \mathcal{M}^{\star})$
4 $\mathcal{C} \leftarrow \textsc{Retrieve}(\mathcal{M}_{\text{scope}}, q)$
5 $\mathcal{N} \leftarrow \textsc{Rerank}(\mathcal{C}, q, \rho, \phi)$ `// Eq. 3; soft suppression`
6 $\mathcal{N}_k \leftarrow \textsc{TopK}(\mathcal{N}, k)$
7 **if** $\mathcal{N}_k = \emptyset$ **then**
8 $\tau \leftarrow \textsc{Rollout}(\mathcal{S}, x, \emptyset)$ `// vanilla fallback`
9 **end**
10 **else**
11 $\tau \leftarrow \textsc{Rollout}(\mathcal{S}, x, \mathcal{N}_k)$
12 **end**
13 $R \leftarrow \textsc{Eval}(\text{Env}, x, \tau)$
14 **end**

the inference budget of a bare student. Finally, the environment returns a binary terminal reward $R \in \{0, 1\}$ (line 9), consistent with the sparse-feedback regime used throughout brewing and requiring no test-time labels or oracle solutions.

A.3 Detailed Setting of Benchmarks

Table 4: Original benchmark statistics and our experimental splits. "Brew" is $\mathcal{D}_{\text{train}}$ for note distillation; "Test" is $\mathcal{D}_{\text{test}}$ for serving.

Scenario	Benchmark	Original		Ours	
		Train	Test	Brew	Test
Math	MATH	7,500	5,000	1,000	5,000
Math	GSM8K	7,473	1,319	1,000	1,319
Coding	MBPP	∼500	∼500	257	163
Coding	MBPP+	—	378	—	154
Tool-use	AppWorld `train`	90 (30 scen.)		90	—
Tool-use	AppWorld `test_normal`	168 (56 scen.)		—	168
Tool-use	AppWorld `test_challenge`	417 (139 scen.)		—	417

Table 4 summarizes the original public benchmark sizes and the processed splits used in our experiments.

Math. MATH and GSM8K originally contain 7,500 and 7,473 training problems respectively. For computational efficiency during brewing, we only use the first 1,000 problems from their official `train` splits. For evaluation, we use the complete official `test` splits (5,000 for MATH and 1,319 for GSM8K), reporting accuracy under symbolic equivalence checking.

Coding. We build on the sanitized MBPP benchmark (427 Python tasks), which is a subset of the original MBPP corpus (∼500/500 train/test partition). Following the standard convention, we use tasks with $\text{task_id} \in [11, 510]$ for `train` (257 tasks) and $\text{task_id} > 510$ for `test` (163 tasks). MBPP+ extends MBPP with stronger tests (378 tasks) but provides no explicit train/test split. Thus, we construct a strictly held-out test set by filtering out all MBPP+ tasks whose IDs overlap with our MBPP `train` split, retaining 154 tasks for evaluation. We report Pass@1 using native assertions for MBPP and EvalPlus tests for MBPP+.

Tool-use. AppWorld originally provides 729 interactive API tasks partitioned into `train` (90 tasks), `dev` (57), `test_normal` (168), and `test_challenge` (417). We directly brew on the official `train` split and evaluate on the two official test splits of increasing difficulty: `test_normal` and `test_challenge`. We report Task Goal Completion (TGC) over tasks and Scenario Goal Completion (SGC).

A.4 Qualitative Example: Why Brewed Notes Generalize

A natural concern is whether notes validated by recovery on a specific brew task encode task-specific patches rather than transferable knowledge. The structured note schema is explicitly designed to prevent this: fields such as `trigger_pattern`, `corrective_rule`, and `not_applicable_when` constrain the teacher to describe *when* and *how* to apply a rule, not to memorize answers. Below is a representative note brewed from a MATH training problem involving factorial simplification.

This note encodes a general algebraic strategy without storing any final numeric answer—the brew task's correct result ($\sqrt{576} = 24$) is absent. Instead, it captures the abstract corrective rule (simplify before computing), its precondition (shared factors exist), and the failure signature that should trigger retrieval (premature full computation). The `not_applicable_when` field further constrains the note's scope, preventing retrieval on tasks where the rule does not apply. Quantitatively, this single note was retrieved and used on **1,200 distinct test tasks** with a **55.6% success rate** when applied, confirming that the structured schema successfully distills reusable heuristics rather than task-specific patches.

Example Brewed Note (MATH, factorial simplification)

[Trigger] *When should this note be retrieved?*

The student computes the full factorial before simplifying the expression.

[Corrective rule] *What should the student do instead?*

Simplify the expression algebraically before performing any calculations to reduce computational complexity and avoid errors.

[Minimal pattern] *How to execute, step by step.*

Cancel common factors in the numerator and denominator before calculating the full value of the factorial.

[Anti-pattern] *What behavior to avoid.*

Directly computing the full factorial without simplifying the expression first, which increases the risk of computational errors and unnecessary complexity.

[Not applicable when] *When to suppress this note.*

When the denominator does not share any common factors with the numerator or when simplification is not possible.

[Evidence] *The failure trajectory that triggered this note.*

The student computed $8! = 40320$*, divided it by 70 to get 576, and then took the square root to arrive at 24, but incorrectly claimed the result was 64 and 8, suggesting a misunderstanding of the computation steps.*

A.5 Real-World Deployment: A Case Study on Terminal-Bench

To complement the controlled academic benchmarks above, we deploy AgentBrew on **Terminal-Bench 2.0** (Merrill et al., 2026), a benchmark comprising 89 real-world terminal tasks spanning 16 categories including software engineering, system administration, data science, security, and scientific computing. We use a stratified 25/64 train/test split and integrate AgentBrew into the Claude Code (Anthropic, 2025) agent harness, with `deepseek-v4-pro` (Xu et al., 2026) as the teacher and `deepseek-v4-flash` as the student. The full pipeline mirrors our main experiments: student baseline, teacher baseline, per-task brewing, category-level curation, Ralph validation, and final serve.

Main Results. Figure 4 summarizes the key findings. The standalone student achieves only 45.3% pass rate, while the teacher reaches 50.0%. With AgentBrew-brewed skills, the student climbs to **51.6%**, surpassing the stronger teacher while costing $0.04 per task—a 33% reduction from the

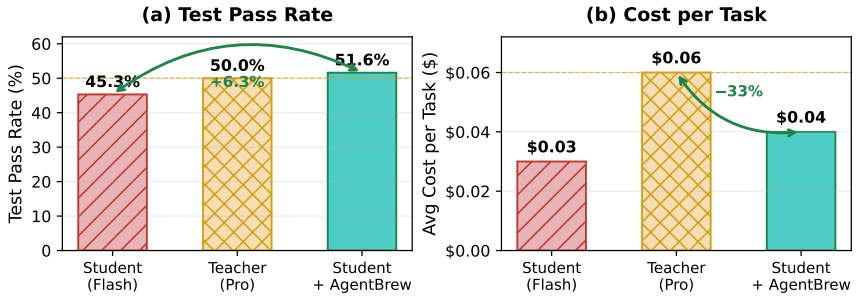


Figure 4: Test pass rate and cost per task on Terminal-Bench 2.0 (64 test tasks). (a) **Left:** pass rate comparison. AgentBrew surpasses the teacher model (51.6% vs. 50.0%) while starting from a substantially weaker student (45.3%). (b) **Right:** average cost per task. AgentBrew costs \$0.04/task—33% less than the teacher's \$0.06/task. Dashed lines mark the teacher's reference level.

teacher's \$0.06. The one-time brewing cost across all training tasks totals approximately \$116, which is amortized over unlimited downstream deployments.

Key Findings. This case study demonstrates three practical properties of AgentBrew beyond the controlled benchmarks in prior subsections. *First*, the method generalizes to open-ended, real-world agent tasks spanning 16 qualitatively distinct categories—not just math, coding, and tool-use. *Second*, the cost advantage is realized under actual API pricing: the student+AgentBrew combination delivers teacher-surpassing quality at roughly two-thirds the cost of the teacher alone. *Third*, the skill injection overhead is negligible at test time: on the 23 "easy" tasks where all three configurations pass, AgentBrew costs \$0.01 per task—identical to the bare student—indicating that retrieved skills do not inflate token consumption on tasks the student already handles well. These findings corroborate our earlier analyses: the Ralph Loop's student-side validation ensures stored knowledge is empirically effective, and student-aware synthesis prevents the teacher from producing guidance the weak executor cannot follow.

A.6 Scaling with Training Data

We further ask whether AgentBrew exhibits *scaling*: as more training trajectories are brewed, does test-time performance improve? We incrementally brew on subsets of MATH $\mathcal{D}_{\text{train}}$ containing 500, 1000, 1500, and 2000 samples, respectively, while keeping all other pipeline settings fixed. For each checkpoint, we freeze the resulting memory $\mathcal{M}^{\star}$ and evaluate it with Qwen3-14B in a single test-time rollout. To control evaluation cost, we report accuracy on a fixed subset of 2000 sampled MATH test problems rather than the full `test` split.

Figure 5 summarizes the scaling trend. The number of brewed notes grows from 21 to 166 as training exposure increases, indicating that AgentBrew continues to distill and retain useful knowledge rather than saturating early. Test accuracy rises from 74.15% at 500 training problems to 76.15% at 2000, with a modest dip at 1000 problems (73.85%) before recovering at larger scales. Overall, the upward trend at 1500–2000 problems suggests that AgentBrew benefits from additional brewing data on MATH.

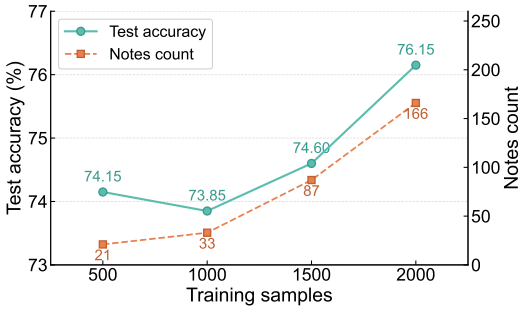


Figure 5: Scaling on MATH benchmark.