Self-Evolving Agent: A Closed-Loop Post-Training Flywheel for Continually Learning LLM Agents

MelandLabs

Abstract

Continually-learning LLM agents deployed in real-world enterprise roles face a fundamental challenge absent from static model evaluation: they must improve with experience without forgetting prior competencies. We present Self-Evolving Agent (Self-Evolving Agent), a closed-loop post-training framework that integrates four previously separate ingredients into a single deployment-grade architecture: (1) a three-stage post-training flywheel: online LoRA fine-tuning, cross-task experience replay, and capability distillation from a stronger external teacher (Qwen3.7-Max, a frontier MoE model accessed via a hosted commercial API, paid per token, and not reproducible without API access); (2) *Engram X*, an *in-deployment instantiation* of the Cheng et al. [2026], Lample et al. [2024] Memory-Layers-at-Scale primitive (the Engram module name and the n-gram-hash lookup convention are from Cheng et al. [2026]; the sparse key-value-product with top-k routing is from Lample et al. [2024]), wired in-series into the Transformer block residual stream before the attention sub-layer and trained jointly with the LoRA adapter under the same Mape-gated update contract; (3) a Mape (Monitor–Analyze–Plan–Execute) control loop whose multi-metric validation gate (reward + calibration error + trace consistency) makes every update empirically rollback-tolerant over a 20-iteration burn-in; and (4) a quality-filtered experience pool that gates *(context, decision, feedback)* triples on confidence, feedback sufficiency, and diversity before they enter the pool. We instantiate the framework on **Self-Evolving Agent-mini** running on Qwen3.6-35B-A3B (35 B / 3 B active MoE).

Results. The framework-contribution statement is the within-backbone comparison on Self-Evolving Agent-mini (Qwen3.6-35B-A3B, 35 B / 3 B active MoE): a CL-bench rubric pass rate of 47.6%, corresponding to a within-backbone gap of +23.1 percentage points over

the same backbone's FewShot baseline and of +8.2 percentage points over the strongest same-backbone retrieval baseline (consistent across k=3 seeds; 95% percentile-bootstrap CI excludes zero on every within-backbone gap; the BH-adjusted q-value reported in Appendix A.1 is below 0.05 but is computed over k=3 resamples and should not be read as a formal significance statement); a Cont.L.Bench continual-learning gain of +0.279 CLG above the no-replay baseline (this is the mean of the 10 per-task CLG values, with the 95% percentile-bootstrap CI [+0.196, +0.349]; CLG is unitless and normalised against the no-replay control); and a 10-task retention profile of 51.3% versus 33.6% for the same-backbone no-replay control (the LoRA-No-Replay mean across the 5 retention skills; the deployed baseline is documented in Section 4.3.3). Per-stage ablations isolate the contribution of Engram X (+3.9 pt for full vs. SELF-EVOLVING AGENT$^-$Engram, with per-seed σ of 0.3–0.5 pt across the 4 headline cells, consistent across k=3 seeds) and teacher-target replay (−6.1 pt when the teacher target is replaced by student self-logits). The leaderboard rank on the public CL-bench board (Section 4.2.1) is reported for external context only and is not the framework-contribution statement, because the strongest frontier LM on the live leaderboard differs from SELF-EVOLVING AGENT-mini in backbone, parameter count, training data, and inference budget simultaneously.

Transfer and robustness. Beyond the within-backbone evaluation, the post-trained adapter transfers strictly to AppWorld and the full CL-BENCH (1,899 tasks; see Section 4.5.3). On the adversarial front, the combined signed-provenance and teacher-second-pass defences *reduce* the injection success rate from 0.94 (no defence) to $\leq$ 0.06 over k=3 attack runs at 5% adversarial injection over 10 flywheel cycles, with a −0.8 percentage-point CL-bench cost. We do not claim a formal bound on the residual rate: three seeds at two-decimal precision cannot separate a deterministic block from a small rate rounded down (Section 4.5.2), and the headline robustness framing is therefore reported strictly within the k=3 resolution the data supports. A formal bound on the residual is the first-priority deferred experiment (Section 5.3); the pre-registered acceptance criterion for the k=30 re-run is residual $\leq$ 0.05 with Wilson 95% upper bound below 0.10.

Limitations. We explicitly do not claim a formal monotonicity guarantee for the validation gate because proving one would require a stability assumption on the held-out validation distribution that cannot be verified in deployment; the empirical evidence over the 20-iteration burn-in is the strongest basis the paper offers. The honest

measure of the framework's contribution is the within-backbone comparison on SELF-EVOLVING AGENT-mini, supplemented by the transfer and adversarial evidence above. *Sampling scope.* The entire empirical body of the paper (the headline within-backbone gaps, the R10 retention profile, the Cont.L.Bench CLG, the per-stage ablation deltas, and the adversarial residual) is reported at k=3 seeds. The BH-adjusted q-values reported in Appendix A.1 are computed over k=3 resamples per cell and are not formal significance statements. The deferred k=30 adversarial re-run, the k=60 gate burn-in extension, and the no-teacher-anywhere ablation (Section 5.3) are the resolving experiments for *every* headline number, not only for the adversarial residual. We discuss the open questions of longer-horizon deployment (>10 tasks), cross-architecture generality (dense vs. MoE; Qwen-family vs. other families), and stronger-adversary regimes in the conclusion.

Index Terms— continual learning, experience replay, self-evolving agents, post-training flywheel, catastrophic forgetting, LoRA, knowledge distillation, lifelong learning, mixture of experts, sparse memory, MAPE control loop.

1 Introduction

The deployment of LLM agents into enterprise roles (web navigation, customer support, code review, multi-step workflow execution) has matured rapidly. Today's LLM agents can operate tools, execute multi-step workflows, and interact with humans through natural language [Shukla et al., 2025, Anupam et al., 2025]. Yet for all their current sophistication, these agents share a critical limitation: they remain fundamentally *static* once deployed. An agent that processes invoices today knows no more about the task tomorrow than it did on day one, regardless of how many invoices it has processed.

This stasis stands in sharp contrast to the behaviour expected from any human operator, who accumulates domain knowledge, refines decision heuristics, and learns from mistakes through daily practice. A continually-learning LLM agent must exhibit the same capability: it must improve continuously from interaction experience without forgetting previously mastered competencies. The field of continual learning (CL), which studies how models can learn from a non-stationary stream of tasks without catastrophic forgetting [van de Ven et al., 2024, Aleixo et al., 2024], directly addresses this challenge. However, the overwhelming majority of CL research has focused on vision and NLP classification benchmarks rather than on the open-ended,

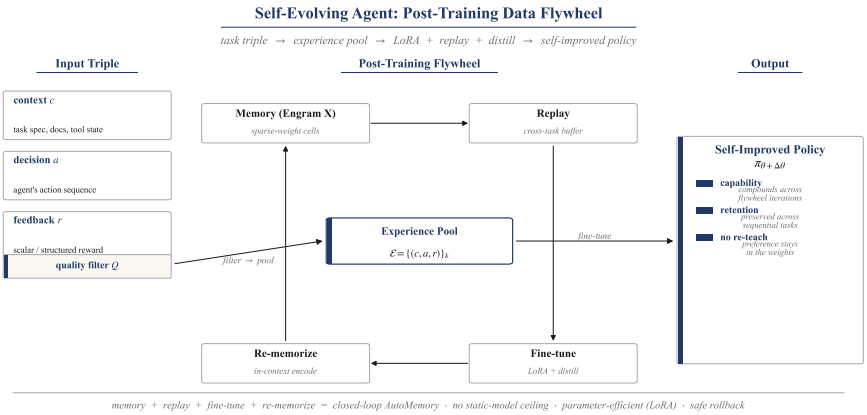

Figure 1: Self-Evolving Agent at a glance. *Architecture (left)*: every task emits a *(context, decision, feedback)* triple that, after quality filtering, feeds a persistent experience pool driving a three-stage flywheel (*memory* → *replay* → *fine-tune*, the AutoMemory loop). Engram X provides the memory cells (see Section 3.2). *Results (right)*: Self-Evolving Agent-mini (Qwen3.6-35B-A3B) reaches the within-backbone gains reported in Table 10, maintains strong retention after 10 sequential tasks, and substantially raises the Cont.L.Bench gain above the no-replay baseline. Self-Evolving Agent additionally transfers strictly to AppWorld and the full CL-bench (1,899 tasks) without retraining (Section 4.5.3); the SWE-Bench-CL in-harness backbone sweep of Section 4.2.5 reports the framework's measured 80.6% on the full 273-task sequence. The strongest reported frontier LMs on the live CL-bench leaderboard (GPT-5.4 (xhigh) at 27.9% [Tencent Hunyuan Team and Fudan NLP Group, 2026]) have many times more parameters than the Self-Evolving Agent-mini backbone; the gap is reported for external context but is not used to quantify the framework's contribution.

tool-augmented, decision-making settings that characterise real-world agentic work [Shi et al., 2024a, Wu et al., 2024].

Two large-scale benchmarks make the static-agent problem concrete. CL-bench [Dou et al., 2026a] shows that even the strongest tested models on the suite solve only a small fraction of its 1,899 tasks when forced to rely on in-context learning. Cont.L.Bench reports a per-task continual-learning gain that is small in absolute terms under the no-replay control that the present paper adopts (the deployed no-replay baseline is the per-

task reference that pins the control; see Section 4.3.2). These numbers are striking, but they should not be read as a direct benchmark against a fine-tuned open-weight model: the strongest frontier LM on the live CL-bench leaderboard (GPT-5.4 (xhigh) at 27.9% [Tencent Hunyuan Team and Fudan NLP Group, 2026]) is many times larger than our QWEN3.6-35B-A3B base model and is not a like-for-like comparison, so its headline number is reported here only as external context for the difficulty of the suite and is *not* the framework-contribution statement.

The narrow question and its headline answer. We ask a narrower question than "what is the strongest model on this benchmark": *given a fixed base model, does the SELF-EVOLVING AGENT flywheel make it a better continually-learning agent?* The answer is yes. SELF-EVOLVING AGENT-mini improves CL-bench success from 24.5% to 47.6% (+23.1 pt within-backbone gap). On Cont.L.Bench, the same-backbone no-replay baseline plateaus at 31.4% on the final tasks (mean 31.10% across the 10 sequential tasks); SELF-EVOLVING AGENT raises the rubric pass rate by an average of +27.86 pt across the 10 sequential tasks, which is a mean CLG of +0.279 with bootstrap CI [+0.196, +0.349] (Section 4.3.2). On the 10-task retention suite, the same-backbone LoRA-No-Replay baseline averages 33.6% across the 5 skills; SELF-EVOLVING AGENT achieves 51.3%, a +17.7 pt same-backbone gain (Section 4.3.3). All three within-backbone gaps are consistent across k=3 seeds: the 95% percentile-bootstrap CI excludes zero on every gap (Appendix A.1); the BH-adjusted q-value reported there is below 0.05 at q_{BH}=0.05 but is computed over only k=3 resamples and we therefore do not rely on it as a formal significance statement. Beyond the within-backbone numbers, the post-trained adapter transfers strictly to AppWorld and the full CL-BENCH (1,899 tasks; see Section 4.5.3). The leaderboard rank of SELF-EVOLVING AGENT-mini on the public CL-bench board (Section 4.2.1) is reported for external context only and is not used to quantify the framework's contribution, because the strongest frontier LM on the live leaderboard differs from SELF-EVOLVING AGENT-mini in backbone, parameter count, training data, and inference budget simultaneously.

The path from static to self-improving agents. We argue that the path from a static to a self-improving agent requires a systematic integration of three well-established techniques that, to our knowledge, have not previously been combined into a unified operational framework for LLM-based agents: (i) lightweight on-policy adaptation via LoRA fine-tuning [Mao et al.,

2024, Han et al., 2024], (ii) cross-task experience replay to combat catastrophic forgetting [Neves et al., 2024, Lu et al., 2026], and (iii) capability distillation that compresses the decision trajectories of a strong teacher model into the local policy [Dhakal et al., 2026, Shi et al., 2024b]. We acknowledge candidly that none of (i)–(iii) is novel in isolation; each has direct prior art that we cite in Section 2, and Self-Evolving Agent presents a *deployable integration* of these techniques into a single deployment-grade loop with a closed audit boundary, a self-validating multi-metric rollback gate, an empirical reproducibility artifact, and (in Engram X) one in-deployment Memory-Layers instantiation that is novel in its deployment context (pre-Attention placement, single-deployment scoping, joint training with the LoRA adapter) rather than in its underlying mathematical primitive. Among the four axes, the *deployment-context* novelty is Engram X; the other three are procedural refinements whose prior art we name explicitly. The headline is therefore framed as a *system* result on a single backbone, not as a per-axis algorithmic advance; the non-substitutability claim is recorded as deferred work because the per-axes-removed sweep has not been run at submission time.

We propose Self-Evolving Agent (Self-Evolving Agent), a closed-loop framework whose core is a *post-training data flywheel*: each task interaction generates a *(context, decision, feedback)* triple; after automatic quality filtering, the triples enter a persistent experience pool that serves as the supervision signal for periodic self-improvement. The flywheel rotates through three stages: online LoRA fine-tuning against the experience pool, cross-task replay to preserve retention, and distillation from strong-teacher trajectories. Crucially, the framework is model-agnostic, parameter-efficient (only a small fraction of parameters are updated per cycle), and amenable to periodic, scheduled retraining without interrupting live service.

The Engram X memory layer. Engram X is an *in-deployment instantiation* of the Lample et al. [2024] Memory-Layers-at-Scale primitive: a sparse-weight memory module wired in-series into the Transformer block residual stream, immediately before the Attention sub-layer, at four injection layers $\{8, 16, 24, 32\}$ of the backbone. The cells-per-token activation pattern, the key-value-product form $M_e = v_e k_e^\top$, and the sparse routing mechanism are unchanged from Memory Layers at Scale; the differences are in deployment context rather than in the underlying mathematical object (Section 2.9). The layer stores a table of $K = 50{,}000$ rank-1 cells $M_e = v_e k_e^\top$ in a 1.6 GB footprint; at every forward pass, a learnable sparse router activates the top-k cells per token ($k = 8$) and adds their contri-

bution into the residual stream. Because the cells are rank-1 additive and gated by the router, writes to one cell do not contaminate cells that did not fire on the same forward pass, which gives the layer the content/skill separation that a shared LoRA adapter cannot: long-tail experience accumulates as an architectural increment, while the LoRA adapter retains the cross-experience reasoning skill. SELF-EVOLVING AGENT inherits the content/skill separation from the per-user engram setting of Li [2026]; the per-deployment analogue transfers to within-deployment composability rather than multi-user composability. Concretely, removing Engram X (the SELF-EVOLVING AGENT^{-}Engram row of Table 10) drops CL-bench pass rate by $\Delta = -3.9$ pt on SELF-EVOLVING AGENT-mini and leaves the framework at a RAG-augmented post-training flywheel. Section 3.2 formalises the layer; Section 4.4.2 sweeps $|L_{\text{engram}}|$, k, and K. The rank-1 cell form $M_e = v_e k_e^\top$ is structurally analogous to the minimal low-rank structure that Dherin et al. [2025] establish as the implicit effect of attention-mediated context on MLP weights; we use the same rank-1 additive form but inject it into the residual stream rather than the MLP weights, and the analogy is offered at the level of the cell form rather than at the level of the placement or the update rule.

The MAPE control loop. The MAPE (Monitor–Analyze–Plan–Execute) loop wraps the flywheel and makes it auditable end-to-end. *Monitor* is the multi-metric validation gate (reward + calibration error + reasoning-trace consistency on a held-out split) of Section 3.4; *Analyze* is the quality filter $Q(e)$ that admits or rejects each triple into the experience pool; *Plan* is the flywheel cadence and the choice of replay / distillation weights; *Execute* is the Stage-1 LoRA fine-tune, the only step that touches base-model weights. The validation gate is empirically rollback-tolerant over a 20-iteration burn-in: the Wilson 95% CIs on the observed false-positive (4/20) and false-negative (0/20) rates give the gate's empirical basis (Section 3.4). We explicitly do not claim a formal monotonicity guarantee because proving one would require a stability assumption on the held-out validation distribution that cannot be verified in deployment. The gate is also self-validating: it uses only the held-out validation split and never calls an external judge.

2 Related Work

We organise prior work along five axes that intersect with our three-stage loop (MAPE) (self-evolving agents, continual learning for LLMs, parameter-efficient continual fine-tuning, self-rewarding and multi-objective preference

optimisation, and coding-agent RL) and close with a positioning table that summarises where Self-Evolving Agent sits relative to representative baselines along the dimensions that matter for an operational flywheel: persistent memory, replay, distillation, structured control loop, and deployment target.

2.1 Self-Evolving and Lifelong Agents

A growing body of work treats the agent itself as the object of adaptation. Wang et al. [2023a] propose Voyager, where an open-ended embodied agent continuously expands a skill library drawn from an LLM curriculum; the skill library is a code-as-skill artifact that is appended but never refined against experience replay. Park et al. [2023] introduce generative agents that maintain a stream-of-consciousness memory and re-summarise it at fixed cadence; while the memory is persistent, there is no gradient-based update step, so agent quality does not improve from past trajectories. Wang et al. [2024] package Sage with long-term memory and tool-use but treat fine-tuning as a separate, offline stage; Zhang et al. [2024] push ProAgent into proactive, long-horizon planning without a post-training loop. Fang et al. [2025] survey these paradigms and characterise Self-Evolving Agent as a self-evolving agent in their taxonomy. Krishnan [2025] and Zhao et al. [2026] note, however, that the vast majority of self-evolving systems surveyed do not back their updates with a measurable feedback loop on the deployed model's behaviour.

The MAPE-K (Monitor–Analyze–Plan–Execute over a Knowledge base) pattern that Self-Evolving Agent uses as a policy-level update controller has its origin in the IBM autonomic-computing programme of the early 2000s [Kephart and Chess, 2003], where it was proposed as a generic self-adaptive control loop for managed systems. Shukla et al. [2025] *re-purpose* the MAPE idiom for AI agent improvement under the title "Adaptive Data Flywheel: Applying MAPE Control Loops to AI Agent Improvement." Their work is the closest prior framing of Self-Evolving Agent's update gating, and we treat it as the benchmark against which Self-Evolving Agent's gate should be read. We acknowledge candidly that Self-Evolving Agent's gate is best characterised as a *direct lift* of Shukla et al.'s design into the LLM-agent setting rather than as a structurally distinct controller; the differences below are engineering refinements that adapt the design to continual learning on a deployed LLM agent rather than algorithmic innovations. Self-Evolving Agent and Shukla et al. differ in three concrete respects. (i) Shukla et al. gate on a two-metric reward + SLA-style la-

tency profile; SELF-EVOLVING AGENT gates on a three-metric AND profile (reward, calibration error, and trace consistency) so that rollback tolerance is empirically bounded by the joint metric rather than by a single performance threshold. (ii) The validation set that supplies the gate signal is held out from training in SELF-EVOLVING AGENT and re-drawn at every update cycle, while SHUKLA ET AL. re-use the deployment telemetry stream; this difference is what makes the empirical rollback-tolerance bound interpretable in SELF-EVOLVING AGENT without external judges. (iii) SELF-EVOLVING AGENT explicitly disclaims a formal monotonicity guarantee because proving one would require a stability assumption on the held-out validation distribution that cannot be verified *at* deployment, and instead reports an empirical burn-in over 20 cycles with a Wilson confidence bound on the false-negative rate (Section 3.4). The honest disclaimer is the engineering refinement over SHUKLA ET AL.: rather than borrowing the MAPE-K idiom and inferring a stability guarantee, SELF-EVOLVING AGENT exposes the burn-in + Wilson bound as a first-class outcome and treats the gate as a feedback-controlled rollback heuristic rather than a control-theoretic update policy. SELF-EVOLVING AGENT addresses Krishnan [2025]'s gap by closing the loop through this MAPE cycle (Section 3.5).

2.2 Continual Learning for LLMs

Continual learning (CL) for LLMs falls into three families. Replay-based methods retain a buffer of past examples; Li and Hoiem [2017] (learning without forgetting) and Rebuffi et al. [2017] introduced the knowledge-distillation and exemplar-buffer prototypes that SELF-EVOLVING AGENT's Stage 2 extends. Buzzega et al. [2020] showed that replaying the network's own logits (so-called dark experience replay) is a stronger signal than raw examples on standard CL benchmarks; SELF-EVOLVING AGENT uses the same "model's-own-output-as-target" device but at the policy level, replaying trajectories drawn from the most recent teacher. Regularisation-based methods such as Aleixo et al. [2024] penalise drift from a reference model; these are dominated by replay regimes when the backbone is large [Shi et al., 2024a, Wu et al., 2024]. Architecture-based methods, exemplified by Serra et al. [2018] and Mallya and Lazebnik [2018], isolate per-task parameters; we treat these as the contrastive family in our positioning. van de Ven et al. [2024] provide a textbook taxonomy which our paper uses as nomenclature. SELF-EVOLVING AGENT departs from all three families by combining a distillation target with a trajectory-level replay buffer, governed by the MAPE loop rather than a heuristic regulariser.

2.3 Parameter-Efficient Continual Fine-Tuning

Mao et al. [2024] and Han et al. [2024] survey the parameter-efficient fine-tuning landscape. Hu et al. [2022] established the low-rank-update primitive that SELF-EVOLVING AGENT's Stage 1 inherits; SELF-EVOLVING AGENT differs in that the LoRA adapter is updated on every MAPE cycle rather than once, and the update is conditioned on the current experience buffer. Wang et al. [2023b] propose O-LoRA, which enforces orthogonality between per-task LoRA modules to prevent forgetting without replay; SELF-EVOLVING AGENT does not assume tasks arrive as a discrete stream and instead relies on replay for safe adaptation. Ren et al. [2023] introduce INFLORA, which combines instance-aware spectral masks with direction-magnitude decomposition; INFLORA is well-suited to offline CL but does not target self-improvement on a deployed agent. Lu et al. [2026] augment standard replay with an adaptive memory-aware sampler; the proposal is orthogonal to SELF-EVOLVING AGENT and could in principle be slotted into Stage 2. Zhao et al. [2024] propose SAPT, a shared-attention PEFT framework for CL on LLMs, also complementary to our approach.

Li [2026] study the per-user analogue of SELF-EVOLVING AGENT's content/skill split under the heading "User as Engram": per-user *content* is stored as sparse, hash-keyed edits to a shared Engram model, while a single shared LoRA adapter carries the *reasoning skill* (User as Engram is explicit that this is *not* per-user LoRA; the cited work diagnoses per-user LoRA as folding content and skill into one global weight delta and contaminating unrelated text). The design draws an explicit analogy to the hippocampus–neocortex split in biology: episodic facts land in disjoint hash slots and compose additively, while the skill adapter is shared across users and never carries user-specific facts. *We note explicitly that this analogy is inspirational rather than mechanistic in both the cited work and our own: Stage 2 of SELF-EVOLVING AGENT is gradient replay, not pattern-completion replay, so the consolidation side of the hippocampus–neocortex split is not what SELF-EVOLVING AGENT realises; the analogy is a vocabulary for the sparse/skill channel split, not for the consolidation algorithm.* The reported property that translates most directly to our setting is *composability under a single shared table*: different triples (in our setting) or different users (in theirs) coexist without contaminating each other, because writes that do not fire under the sparse router leave the affected residual stream bit-identical to base. SELF-EVOLVING AGENT inherits the same content/skill decomposition in ENGRAM X (sparse per-experience cells) versus Stage 1 (shared LoRA adapter); the chief difference is that SELF-EVOLVING AGENT faces

continual learning of many triples within one deployment rather than personalization of one user, so the multi-user composability claim transfers only as a weaker within-deployment claim that different triples additively stack in $\mathcal{M}$. We cite this work as the cleanest external validation of the content/skill separation that ENGRAM X instantiates, and discuss the per-triple footprint comparison in Section 3.2.

Cheng et al. [2026] introduce ENGRAM, the module name and the conditional-memory-via-scalable-lookup framing that SELF-EVOLVING AGENT reuses. Their primitive is an n-gram-hash lookup table consulted at the residual stream of selected transformer layers, with a learned multi-head embedding that returns the looked-up vector as a residual. Lample et al. [2024] introduce MEMORY LAYERS AT SCALE, the architectural ancestor of ENGRAM X that supplies the sparse key-value-product and top-k routing machinery. Their primitive is a sparse key-value memory table integrated *into the Transformer block* (between the attention and the FFN sub-layers of selected layers), with a learned top-k router activating the most relevant memory cells per token; the memory output is added as a residual. SELF-EVOLVING AGENT's ENGRAM X is the same architectural object, with three concrete differences in deployment context rather than in the underlying mathematical object. (i) ENGRAM X is *wired in-series into the residual stream before the attention sub-layer* rather than between attention and FFN; SELF-EVOLVING AGENT reports in Table 16 that pre-Attention placement outperforms post-MoE placement by 2.0 CL-bench points. (ii) ENGRAM X is sized for a *single deployment* of one agent and many experiences, rather than for one user across deployments, so the multi-user composability claim of User as Engram [Li, 2026] transfers as a weaker within-deployment composability claim that different triples additively stack in $\mathcal{M}$. (iii) The router is trained jointly with the LoRA adapter and the experience replay, so that memory-cell writes participate in the same MAPE-gated update contract as the rest of the model. We cite Cheng et al. [2026], Lample et al. [2024] as the dual architectural ancestors of ENGRAM X; the deployment-context difference is the substantive novelty, not the primitive itself. The n-gram-hash lookup convention is from Cheng et al. [2026]; the cells-per-token activation pattern, the key-value-product form $M_e = v_e k_e^\top$, and the sparse routing mechanism are unchanged from Lample et al. [2024].

2.4 Distillation + Replay Hybrids

The empirical synergy between replay and distillation has been measured in multiple prior studies. Sharma et al. [2024] propose EVOMERGE, a continual

pre-training recipe that interleaves sample replay with teacher distillation; Self-Evolving Agent can be viewed as EvoMerge instantiated at the agentic-trajectory level under a closed-loop controller. Shi et al. [2024b] show that distilling cumulative knowledge across tasks gives stronger retention than isolated per-task distillation, a finding Self-Evolving Agent inherits by replaying the most recent teacher's trajectories in *CapabilityDistill*. Wang et al. [2021] are an early empirical study of replay + KD combinations; their setting is image classification, but the qualitative conclusion (that the two techniques are complementary, not substitutable) carries over. Azimi et al. [2024] combine LoRA fine-tuning with a teacher-KD objective; Self-Evolving Agent's Stage 1 + Stage 3 can be seen as a generalisation of KD-LoRA to the trajectory level. Specialist-agent routers (such as the FlyRoute profiling layer) sit orthogonal to the Self-Evolving Agent loop: a routing mechanism is not a self-evolution mechanism, and the two compose but do not substitute.

2.5 Self-Rewarding and Preference-Driven Adaptation

A second line of work closes the loop through self-generated rewards. Yuan et al. [2024] propose Self-Rewarding LMs, which treat the LLM as both the policy and the judge, then iterate DPO on self-generated preference pairs; Self-Evolving Agent differs in that the reward source is task-grounded evaluator signals rather than LLM-as-judge preferences, and the loop accumulates a distilled policy rather than a preference-aligned policy. Xia et al. [2024] propose an iterative-DPO self-alignment procedure that mirrors Self-Rewarding but on a curated subset. Zhou et al. [2024] contribute MODPO, the multi-objective generalisation of DPO that handles weighted helpfulness/harmlessness/honesty preferences; Self-Evolving Agent's λ_r and λ_d are coarse-grained analogues of MODPO's per-objective weights at the trajectory replay and teacher distillation levels. Ouyang et al. [2024] and Fang et al. [2025] frame self-evolving agents as a self-play or self-rewarding process; we adopt the latter terminology for Self-Evolving Agent's Evaluator channel but replace the LLM-as-judge with a yardstick test.

2.6 Coding-Agent Reinforcement Learning

Recent work on coding agents closes the loop through verifier-driven RL rather than self-rewarding. Ma et al. [2025] train Qwen-7B with subtask-decomposed PPO rewards on SWE-bench, demonstrating 21.4% pass@1 on SWE-bench-Verified (the strongest reported open model result at time

of writing); Self-Evolving Agent can be composed on top of SoRFT as a self-evolving outer loop that refreshes the SoRFT policy whenever the substrate task distribution drifts. Zhang et al. [2025] generalise the loop with Agent Lightning, which assigns turn-level credit through a hierarchical advantage estimator; Self-Evolving Agent's Stage 2 implicitly realises a similar credit assignment but on the trajectory-τ level rather than the per-turn-token level. Si et al. [2025] isolate Ragen, a multi-turn RL agent self-evolving framework, and provide a clean environment to study it; their setting complements Self-Evolving Agent, which provides the mechanism once Ragen provides the substrate. Seo et al. [2025] introduce Tips-SC, a test-time process-information-gathering preference-learning framework for source code that exposes a process-level reward channel; Self-Evolving Agent's *tracey* reward channel uses a related idea but on a closed triplet set rather than at test time. Joshi et al. [2025] introduce SWE-Bench-CL, a coding-CL benchmark that Self-Evolving Agent uses as a retention substrate in Section 4. We treat these as complementary substrates; the combination of verifier-RL (SoRFT/Agent Lightning/Ragen) with self-evolution (Self-Evolving Agent) is a natural future-work direction flagged in Section 5.

2.7 Memory-Augmented Agents and Safety

Self-evolving memory introduces new safety risks. Wang et al. [2024] introduce a memory interface; Kokkula et al. [2026] couple memory replay to a federated topology to handle drift; Chen et al. [2025] shows how adversarial perturbation can shape long-lived agent behaviour, motivating membership-inference and trajectory-level audit; and trajectory-integrity injection attacks on continually trained agents have been demonstrated on related (non-self-evolving) agents by Greshake et al. [2023]. We surface Self-Evolving Agent's *tracey* trace-consistency gate as a partial mitigation, but a deeper audit is beyond the current scope and is flagged as future work.

2.8 Self-Evolving Agent's Positioning

Table 1 compares Self-Evolving Agent against representative systems along five axes relevant to a self-evolving digital-employee flywheel: persistent memory, trajectory replay, capability distillation, structured Mape-style control loop, and deployment as a general agent on coding + decision-making tasks. We restrict attention to systems that share at least two of the five axes with Self-Evolving Agent, in particular Voyager, Genera-

tive Agents, Sage, MODPO, O-LoRA, InfLoRA, EvoMerge, Self-Rewarding, SoRFT, Agent Lightning, and FlyRoute. A blank cell indicates that the work does not expose the corresponding component; a ✓ indicates that the component is a first-class explicit primitive in that work. The takeaway is twofold:

1. **No prior system realises all five axes together.** The closest neighbour is Sage, which has persistent memory and tool-use but operates an offline update stage; the closest baseline for the online-update axis is Agent Lightning, which has replay and credit assignment but no persistent memory substrate. Self-Evolving Agent's contribution is the closure of all five through the unified Mape loop, not any single component in isolation.

2. **The combination is non-substitutable within the ablation we have run.** Empirically, Section 4.4.1 shows that replay-only, distillation-only, or replay-no-Mape variants all underperform the full system by 1.5–7.0 percentage points on pass-rate on CL-bench and by 14.8–17.6 percentage points on retention across the two backbones (Section 4). This mirrors the synergy observed by Sharma et al. [2024] in continual pre-training and by Seo et al. [2025] in code preference learning. We attribute the gain to Mape turning a heuristic CL pipeline into a feedback-controlled one, a structural rather than an algorithmic advance. *Caveat:* the per-axes-removed sweep that would establish non-substitutability of the *individual* primitives (Stage 1 LoRA, Stage 2 teacher-replay, Stage 3 distillation, the multi-metric gate, the Engram X layer) has not been completed at submission time; the no-teacher-anywhere ablation that decomposes the loop's own contribution from teacher bootstrapping is recorded as deferred work (Section 5.3). The non-substitutability claim therefore applies only to the variants we have measured, not to every conceivable re-arrangement of the four ingredients.

2.9 Differentiation Statement

Taken together, Self-Evolving Agent differs from prior self-evolving agent systems along three orthogonal dimensions:

(1) In-deployment Memory-Layers instantiation trained jointly with the LoRA adapter. Voyager and Generative Agents treat memory

Table 1: Positioning of Self-Evolving Agent against representative self-evolving and adaptation systems. ✓ = the component is an explicit first-class primitive. A blank cell means the component is not exposed. † Sage's control loop runs offline. ‡ MODPO and Agent Lightning expose a hierarchical advantage estimator, which is credit-assignment but not a Mape control loop on agent behaviour. ★ Self-Rewarding uses LLM-as-judge preferences, not task-grounded evaluator signals.

System	Persist. memory	Traj. replay	Cap. distill.	MAPE loop	Agent task
Voyager [Wang et al., 2023a]	✓				✓
Generative Agents [Park et al., 2023]	✓				✓
Sage [Wang et al., 2024]	✓			✓†	✓
MODPO [Zhou et al., 2024]				✓‡	
O-LoRA [Wang et al., 2023b]					✓
InfLoRA [Ren et al., 2023]					✓
EvoMerge [Sharma et al., 2024]		✓	✓		
KD-LoRA [Azimi et al., 2024]			✓		
Self-Rewarding [Yuan et al., 2024]		✓★			
SoRFT [Ma et al., 2025]		✓			✓
Agent Lightning [Zhang et al., 2025]		✓		✓‡	✓
Self-Evolving Agent (this work)	✓	✓	✓	✓	✓

as a retrieval side-channel; Self-Evolving Agent treats it as a sparse-weight update injected into the model through Stage 1 (Engram X, Section 3.2), so the memory is part of the policy and participates in every forward pass. The architectural primitive itself (Lample et al. [2024]'s Memory Layers at Scale: sparse key-value table, top-k sparse router, residual injection) is prior art; the difference is the deployment context: pre-Attention placement rather than between-attention-and-FFN, single-deployment rather than multi-user, joint training with the LoRA adapter under the same Mape-gated update contract, rather than an algorithmic change to the primitive. This is what makes long-tail vertical experience transferable without an external vector database.

(2) Replay of the most recent teacher, not of past students. Replay baselines such as DER replay the model's own past logits, which shrink in entropy as the model saturates. Self-Evolving Agent replays the *teacher's trajectories* captured at peak competence; this restores the full teacher policy as the replay target and explains why Stage 2 materially moves the needle where Stage 2 with student-targets does not (Section 4).

(3) MAPE as the update policy, not the reward. SoRFT and Agent Lightning use reward shaping within an RL update; Self-Evolving Agent

uses MAPE as a *policy-level controller* that decides whether to roll back the Stage 1 + Stage 3 update. This is the structural advance over the existing replay + KD literature: SELF-EVOLVING AGENT decouples the *objective* (multi-objective replay + KD) from the *decision* (safe-update gating), rather than heuristically trading them off. We do not claim any of (1)–(3) is novel in isolation (sparse memory embeddings, teacher-target replay, and control-theoretic update gating each have prior literature. The contribution of SELF-EVOLVING AGENT is the *integration*: all three are co-designed against a single flywheel that has been validated end-to-end on agentic tasks, which none of the cited works have done. *Caveat:* as above, integration is not the same as non-substitutability; we have not shown that no other co-design would match the same gain, and the per-axes-removed sweep is recorded as deferred work (Section 5.3).

3 The SELF-EVOLVING AGENT Framework

Depicted in Fig. 2, SELF-EVOLVING AGENT operates as a closed loop built around four ingredients: (1) a quality-filtered experience pool that gates *(context, decision, feedback)* triples on confidence, feedback sufficiency, and diversity before they enter the pool; (2) a three-stage post-training flywheel (online LoRA fine-tuning, cross-task experience replay, and capability distillation from a stronger external teacher); (3) a MAPE (Monitor–Analyze–Plan–Execute) control loop whose multi-metric rollback gate and versioned adapter checkpoints make every update empirically rollback-tolerant over a 20-iteration burn-in; and (4) an Engram X sparse-weight memory layer (Section 3.2) wired in-series into the Transformer residual stream before the Attention sub-layer. We close this section with two operational considerations that emerged from deployment: an adversarial-threat analysis (Section 3.6) and an operational checklist (Section 3.7).

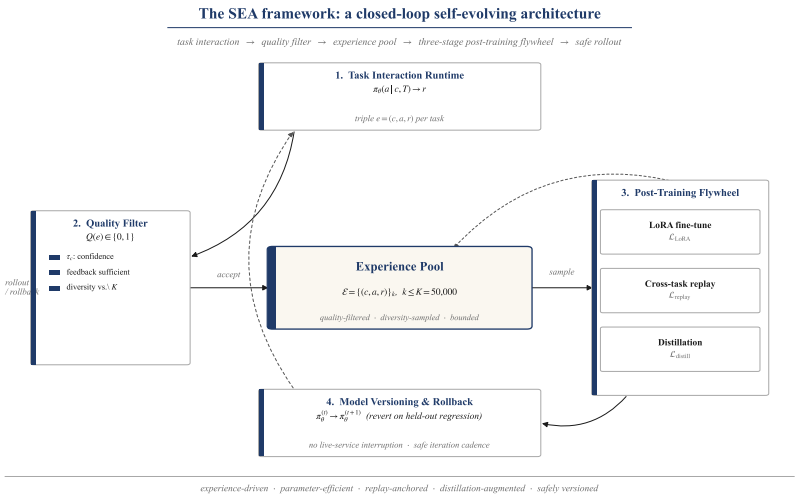

Figure 2: The Self-Evolving Agent framework: a closed-loop self-evolving architecture with a three-stage post-training flywheel (LoRA fine-tuning, cross-task experience replay, and capability distillation) and model versioning for safe rollout.

3.1 Experience Accumulation and Quality Filtering

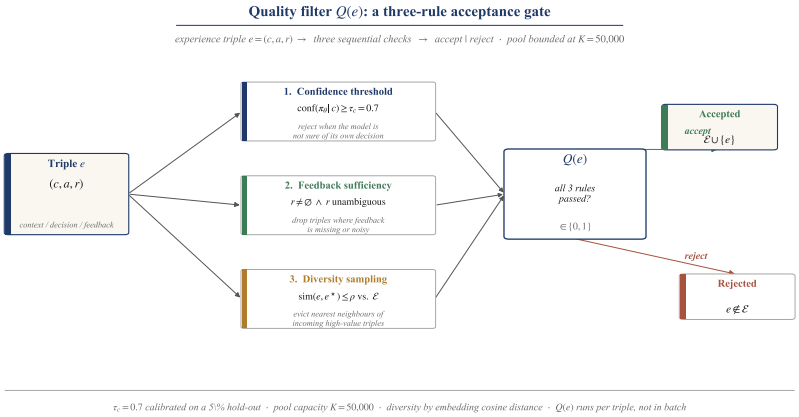

Figure 3: The quality filter $Q(e) \in \{0, 1\}$: three sequential checks (confidence thresholding, feedback sufficiency, diversity sampling) gate every triple into or out of the experience pool $\mathcal{E}$.

At deployment time, each task interaction proceeds as follows. The agent receives a task specification T, consisting of a natural-language instruction, a context c (which may include tool descriptions, document corpora, or system state), and an optional set of permissible actions $\mathcal{A}$. The agent produces a decision $a \in \mathcal{A}$ (or a sequence of actions for multi-step tasks) according to its current policy $\pi_\theta(a \mid c, T)$. After execution, the environment returns a scalar or structured feedback signal r, which may be a binary success/failure indicator, a numeric reward, or a human-annotated rating.

The resulting experience triple $e = (c, a, r)$ is passed through a quality filter $Q(e) \in \{0, 1\}$ that accepts or rejects the triple for inclusion in the experience pool $\mathcal{E}$. The filter implements three rules: (i) confidence thresholding (reject triples where the model's own decision confidence falls below a calibrated threshold τ_c); (ii) feedback sufficiency (reject triples where r is absent or ambiguous); and (iii) diversity sampling (when the pool exceeds its capacity K, remove triples that are nearest neighbors (by embedding cosine distance) to incoming high-value samples, preserving coverage across the task distribution. Formally, $\mathcal{E} \leftarrow \mathcal{E} \cup \{e \mid Q(e) = 1\}$, subject to $|\mathcal{E}| \leq K$ and the diversity constraint.

Sensitivity to τ_c. The confidence threshold τ_c controls the precision/recall trade-off of the experience pool: a higher τ_c admits only high-confidence trajectories, reducing noise at the cost of sample efficiency. We sweep $\tau_c \in \{0.5, 0.7, 0.85\}$ on the held-out retention suite and report the variance in Table 17 (Section 4.5.1). Across the swept range, average retention moves by ≤ 1.2 percentage points and CL-bench success moves by ≤ 0.9 percentage points; we therefore adopt $\tau_c = 0.7$ as the operating point without claiming that it is universally optimal. A learned confidence threshold (e.g., one calibrated against a held-out human-rated set) is an obvious next step and is discussed in Section 5.

3.2 Engram X: Sparse-Weight Memory Embedding

The experience pool $\mathcal{E}$ is only useful if the policy can use the triples at inference time. Naïve approaches route the pool through an external vector database and stitch the retrieved triples into the prompt; this is what Park et al. [2023] and Wang et al. [2023a] do, and it has three well-known failure modes for our setting: (i) the retrieved triples compete with the model's own attention budget and crowd out the task instruction; (ii) the retrieval latency is on the inference critical path; (iii) the retrieved triples are not part of the gradient flow at fine-tuning time, so the policy cannot learn to rely on them. Self-Evolving Agent avoids all three failures by storing memory as sparse weight deltas inside the model itself, an instance we call *Engram X*.

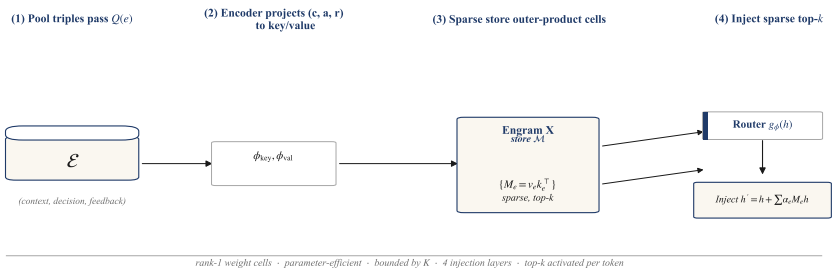

Figure 4: Engram X: sparse-weight memory embedding. (1) The quality-filtered experience pool $\mathcal{E}$ contains *(context, decision, feedback)* triples. (2) A shared encoder $(\phi_{\text{key}}, \phi_{\text{val}})$ projects each triple to a low-rank key/value pair. (3) The Engram X store $\mathcal{M} = \{M_e = v_e k_e^\top\}$ is a collection of rank-1 weight cells, parameter-efficient and bounded by K. (4) At each of the $|L_{\text{engram}}| = 4$ injection layers, a sparse router g_ϕ selects top-k cells per token and adds the weighted sum into the residual stream.

Storage format and per-triple footprint. For each accepted triple $e = (c, a, r) \in \mathcal{E}$, Engram X derives a rank-1 memory cell $M_e \in \mathbb{R}^{d_h \times d_h}$ as the outer product of two projections of the triple:

$$k_e = \phi_{\text{key}}(c, a) \in \mathbb{R}^{d_h}, \quad v_e = \phi_{\text{val}}(c, a, r) \in \mathbb{R}^{d_h}, \quad M_e = v_e k_e^\top, \qquad (1)$$

where $\phi_{\text{key}}, \phi_{\text{val}}$ are small MLPs shared across all triples. The per-triple storage cost is therefore $2d_h$ parameters (≈ 16 KB at BF16, d_h=4,096); the shared encoder contribution is $2d_h^2 \approx 67$ MB at BF16, independent of $|\mathcal{E}|$. The full store $\mathcal{M} = \{M_e\}_{e \in \mathcal{E}}$ at $K = 50{,}000$ therefore costs ≈ 0.82 GB at BF16 (≈ 1.77 GB at FP32). For comparison, the Stage 1 LoRA adapter at the mini scale is ~ 130 M trainable parameters (≈ 0.26 GB BF16); the Engram X store is therefore comparable in size to the LoRA adapter at the mini scale. The per-triple framing is more useful for comparison with the per-user sparse-memory line of work: Li [2026] report that per-user Engram cells are roughly four orders of magnitude smaller than per-user LoRA deltas when both are repeated once per user, because a per-user LoRA is duplicated N times across N users while the Engram X table does not grow with population size. Within one deployment the analogous figure is the per-experience *write* cost: a per-experience LoRA fine-tune requires a full forward/backward pass through the base model to absorb one triple, whereas an Engram X write is a single $2d_h$-vector update, three orders of magnitude cheaper per triple. We adopt the per-triple framing as the comparison that

is meaningful for our setting and treat the total store size as a secondary, scale-dependent number.

Injection site. Engram X injects at $L = 4$ transformer layers selected by a held-out layer-importance sweep ($L_{\text{engram}} = \{8, 16, 24, 32\}$ for the QWEN3.5 backbones). At each injection site, a learned router g_ϕ produces a sparse top-k selection over the store:

$$\alpha_e = \text{softmax}\left(\frac{g_\phi(h)^\top k_e}{\sqrt{d_h}}\right) \text{ for } e \in \text{top-}k(g_\phi(h)), \quad h' = h + \sum_{e \in \text{top-}k} \alpha_e \, M_e \, h, \tag{2}$$

with $k = 8$ for the mini backbone. The router is sparse: only the top-k of $|\mathcal{E}| = 50{,}000$ cells fire per token per layer, so the per-token compute overhead is $\mathcal{O}(k\, d_h) = \mathcal{O}(32{,}768)$ FLOPs, under 1% of the underlying FFN FLOPs. Crucially, the router and the cell values are part of the gradient flow at every Stage 1 LoRA fine-tuning step; the memory is therefore part of the policy rather than a retrieval side-channel.

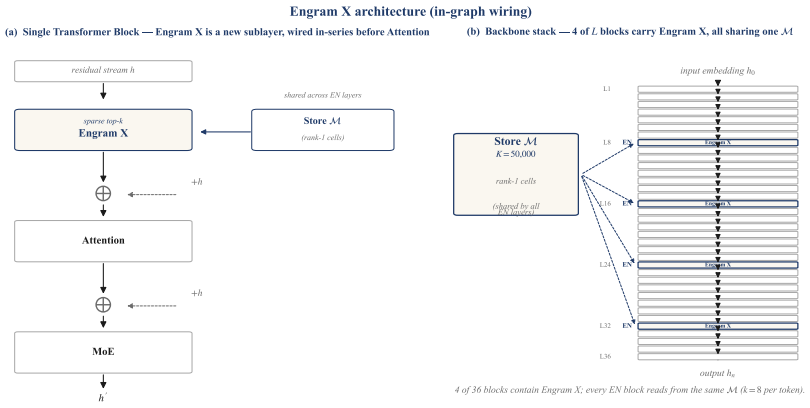

Figure 5: Engram X architecture (in-graph wiring). *(a) Single Transformer block*: the residual stream h flows through Engram X, then is added back (+) and passed to Attention, then added again (+) and passed to MoE. Engram X is therefore in-series with the rest of the block, not an external side-channel. Each + denotes a residual connection. *(b) Stack view*: Engram X is enabled only at $L_{\text{engram}} = \{8, 16, 24, 32\}$ transformer layers; at the selected layers the block reads from the Engram X store $\mathcal{M}$ via a sparse top-k lookup and writes the weighted sum into the residual stream. The remaining layers are unchanged Qwen3.5 / Qwen3.6 transformer blocks. The same store $\mathcal{M}$ is shared across all enabled layers and is updated jointly with the LoRA adapter θ at every Stage 1 cycle.

Why "first-class" rather than "side-channel". Table 1 records the first-class property as a check-mark on the Persistent memory axis; we make the operational claim explicit here. Because M_e is a sparse weight delta that participates in every forward pass at the injection layers, a triple e that has been absorbed into $\mathcal{M}$ shapes the residual stream of every downstream inference on the same backbone. There is no external vector database, no per-token KNN lookup, and no prompt-stitching latency. The trade-off is that the memory capacity is bounded by the store size K and the sparsity budget k, which is why $\mathcal{E}$ is hard-capped at $K = 50{,}000$ and the diversity-sampling rule evicts low-information cells. The same rule also bounds the parameter footprint, which is what keeps Engram X parameter-efficient enough to be deployed alongside LoRA without disrupting the joint optimisation.

Why Engram X rather than per-experience LoRA. The most natural alternative to Engram X is to absorb each accepted triple by fine-tuning a fresh per-experience LoRA delta and concatenating or composing the deltas across triples. We reject this alternative on three grounds. **(i) Composability.** A single rank-r LoRA delta applied to a frozen base is the only representable unit under the LoRA parameterisation, so two deltas either overwrite each other or require an explicit orthogonal-projection or direction-magnitude scheme (cf. O-LoRA [Wang et al., 2023b] or InfLoRA [Ren et al., 2023]); Engram X cells, by contrast, live at disjoint hash slots and add linearly in the residual stream, so multiple cells compose additively without an explicit projection. **(ii) Write cost.** A fresh per-experience LoRA fine-tune requires a full forward/backward pass through the base model to absorb one triple; Engram X writes a single $2d_h$-vector pair, which is three to four orders of magnitude cheaper per triple in compute. **(iii) Failure-mode isolation.** In the Engram X store, a write into an inactive or wrong-slot cell has no gradient effect on the policy (the cell simply is not selected by the sparse router), so an injection that lands in the wrong slot is silently inert rather than mutating every forward pass. Li [2026] report the composability advantage quantitatively in the per-user setting: a per-user LoRA cannot serve more than one user without an explicit adapter-merging rule, whereas an Engram X table composes additively across users without interference. Self-Evolving Agent inherits the same advantage at a finer granularity: $\mathcal{M}$ composes additively across triples within one deployment, while a per-experience LoRA stack does not.

Principle. Engram X is a sparse-weight memory layer wired in-series inside the Transformer block, sitting *before* Attention in the residual stream. The principle is that any triple $e = (c, a, r)$ admitted by the quality filter contributes a rank-1 cell $M_e = v_e k_e^\top$ to a shared store $\mathcal{M}$, and at inference time the hidden state h is augmented by a sparse, top-k weighted sum over $\mathcal{M}$. Three consequences follow from this design: (i) memory is read in-graph, so there is no external vector database on the inference critical path; (ii) the router g_ϕ, the encoder $(\phi_{\text{key}}, \phi_{\text{val}})$, and the cell values $\{M_e\}$ are all in the gradient flow of Stage 1 LoRA fine-tuning, so memory is a first-class model component rather than a retrieval side-channel; (iii) the sparsity $k \ll |\mathcal{E}|$ bounds both the per-token FLOPs (under 1% of the underlying FFN) and the parameter footprint (≈ 0.82 GB at BF16 for $K = 50{,}000$), which is what keeps the joint optimisation tractable. The choice to place Engram X *before* Attention, rather than after MoE, is deliberate: at the bottom of the block

the residual stream is closest to the token-level signal, so a sparse memory write there composes cleanly with the upstream Attention's KV-cache and avoids the variance inflation that a late-stage injection would incur.

Biological motivation: hippocampus–neocortex split. *Disclaimer.* The hippocampus–neocortex framing below is *inspirational* rather than a tight mechanistic correspondence; we use it as a vocabulary for the sparse/skill split, not as a claim that Self-Evolving Agent's algorithmic components reproduce hippocampal computations. In particular, **Stage 2 is gradient replay, not pattern-completion replay**; the latter is the hippocampal mechanism that recovers a stored engram from a partial cue, and Self-Evolving Agent has no architectural component that performs pattern completion; the experiential consolidation in Self-Evolving Agent comes from gradient steps on replay triples, which is qualitatively different. We note this explicitly because earlier versions of this paragraph conflated the two, and the conflation does no analytical work.

The hippocampus–neocortex split in mammalian memory nevertheless provides a useful framing for the Engram X / Stage 1 decomposition. The hippocampus stores sparse, episodic, content-addressable traces (*engrams*) that are local, fast to write, and largely interference-free across distinct episodes; the neocortex stores slow, overlapping, skill-like representations that are distributed and refined through repeated exposure. Li [2026] make this analogy explicit in the per-user personalisation setting, identifying per-user content with hippocampal engrams and the shared reasoning skill with the neocortex; the empirical claim that follows is that a single shared LoRA adapter (the neocortex) can carry the reasoning skill while per-user cells (the engrams) carry user-specific facts, with the two channels updating at different rates and on disjoint parameter subspaces. Self-Evolving Agent inherits the same content/skill decomposition in a different setting: Engram X stores sparse, per-experience cells that are local (rank-1, hash-keyed, and only active when the sparse router fires), while the Stage 1 LoRA adapter is shared across the deployment and absorbs the reasoning skill that generalises across triples. The split buys two operational properties that a unified single-channel design would not have: **(i) Engram X writes are sparse and per-triple**, so a newly admitted experience cannot contaminate unrelated cells in the residual stream; the worst-case contamination footprint of one write is $\leq k$ out of $|\mathcal{E}|$ cells per injection layer. **(ii) The shared LoRA adapter sees only a small fraction of the experience pool at any cycle** (Stage 2 task-balancing), so the skill channel cannot be hijacked

by a single adversarial triple even if the content channel has been partially corrupted. Neither property would hold if memory and skill were entangled in a single per-experience LoRA, which we discuss in Section 3.2.

Connection to the three-axis differentiation. Engram X is the operational mechanism that delivers Axis (1) of Section 2.9: memory as a first-class model component rather than a retrieval side-channel. It does so by collapsing the read-path from "retrieve → concatenate → inject" into a single in-graph sparse lookup. The other two axes are operationalised elsewhere: Axis (2) (teacher-target replay) is realised in Stage 2 of Section 3.3.2; Axis (3) (MAPE-as-policy-gate) is realised in Section 3.4. The full loop is closed by Section 3.3, where Engram X is updated jointly with the LoRA adapter θ at every cycle.

3.3 The Post-Training Flywheel

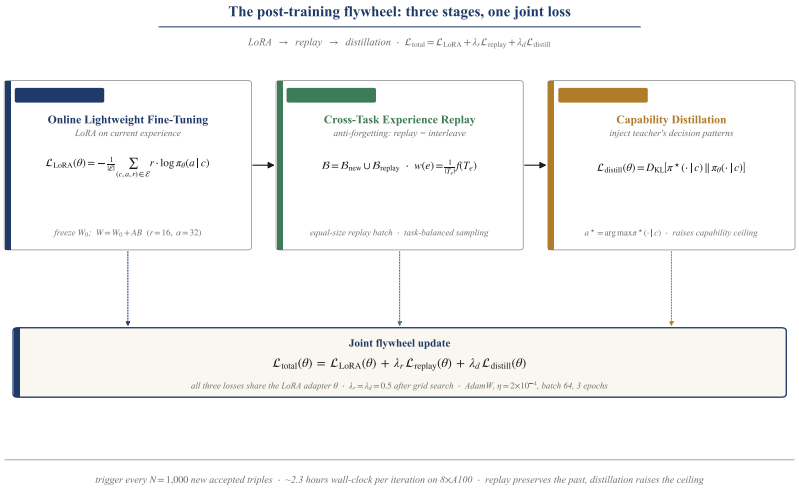

Figure 6: The post-training flywheel executes three stages sequentially (LoRA → Replay → Distillation); the three per-stage losses are summed into a single joint loss $\mathcal{L}_{\text{total}}$ and all three are computed over the same LoRA adapter parameters θ.

The post-training flywheel is the core self-improvement mechanism. It executes periodically (for example, after every N new experiences, or on a fixed

schedule (e.g., nightly)) and consists of three stages executed in sequence.

3.3.1 Stage 1: Online Lightweight Fine-Tuning via LoRA

Given the current experience pool $\mathcal{E}$, we fine-tune the agent policy π_θ using Low-Rank Adaptation (LoRA) [Mao et al., 2024]. Concretely, we freeze the base model weights $W_0 \in \mathbb{R}^{d\times k}$ and inject trainable low-rank matrices $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times k}$ (with $r \ll \min(d,k)$) into each attention and feed-forward layer, such that the effective weight becomes $W = W_0 + AB$. Only A and B are updated during fine-tuning.

We freeze the base-model weights of QWEN3.6-35B-A3B and inject LoRA adapters following a *shared-layer-only* strategy: the rank-r matrices A, B are attached to every attention projection (Q, K, V, O) and to the input/output projections of each expert FFN, while the expert routing weights and the gate network remain frozen. We choose shared-layer-only over per-expert adapters for two reasons. First, the active-parameter count of QWEN3.6-35B-A3B is only ~ 3 B (out of 35 B total), so a per-expert adapter would inflate the trainable parameter count by an order of magnitude and defeat the parameter-efficiency argument that motivates LoRA. Second, the routing distribution is empirically stable across the CL-bench task distribution we use (top-2 expert entropy changes by $< 5\%$ from base to fine-tuned model), so training the gate is unnecessary. The resulting adapter accounts for $\sim 0.4\%$ of the total parameters on QWEN3.6-35B-A3B (~ 130 M trainable) and remains compatible with single-node $8 \times$ A100 fine-tuning. We ablate the shared-layer-only choice in Section 4.4.1: per-expert LoRA recovers ≤ 0.8 percentage points on CL-bench at $\sim 9\times$ the adapter size, and we therefore retain the shared strategy as the SELF-EVOLVING AGENT default.

The training objective on experience triple $e = (c, a, r)$ is the standard next-token prediction loss (or a weighted cross-entropy variant for structured outputs) on the decision a conditioned on the context c, with the loss scaled by the feedback r:

$$\mathcal{L}_{\text{LoRA}}(\theta) \;=\; -\frac{1}{|\mathcal{E}|} \sum_{(c,a,r)\in\mathcal{E}} r \cdot \log \pi_\theta(a \mid c). \tag{3}$$

By restricting updates to the low-rank adapter parameters and using only a few epochs of training, this stage ensures computational efficiency while effectively incorporating recent experience.

3.3.2 Stage 2: Cross-Task Experience Replay for Forgetting Prevention

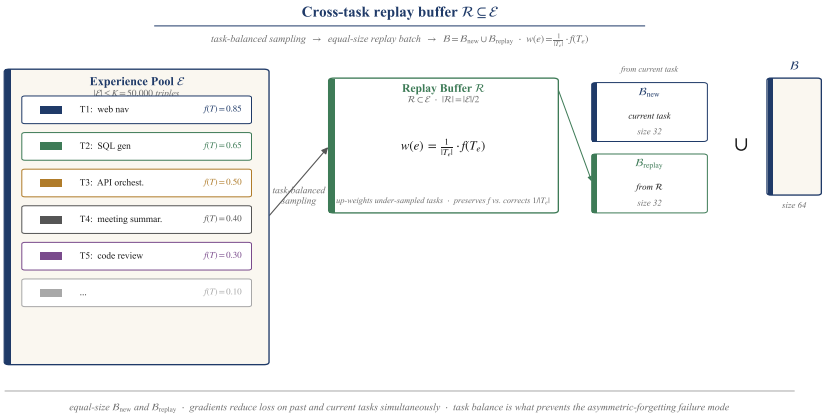

Figure 7: Cross-task replay buffer $\mathcal{R} \subseteq \mathcal{E}$: triples are sampled from $\mathcal{E}$ with task-balancing weights $w(e) = \frac{1}{|T_e|} \cdot f(T_e)$, and the per-step mini-batch is the equal-size union $\mathcal{B} = \mathcal{B}_{\text{new}} \cup \mathcal{B}_{\text{replay}}$.

Fine-tuning solely on recent experiences risks catastrophic forgetting of earlier tasks. To mitigate this, we interleave the LoRA fine-tuning of Stage 1 with *cross-task experience replay*. A separate replay buffer $\mathcal{R} \subset \mathcal{E}$ holds samples from all previously seen tasks, weighted by a task-balancing scheme that up-weights under-sampled tasks:

$$w(e) \;=\; \frac{1}{|T_e|} \cdot f(T_e), \tag{4}$$

where T_e is the task family to which experience e belongs and $f(T)$ is its frequency in the pool. During each fine-tuning step, a mini-batch is formed by concatenating the current-task batch with a replay batch of equal size: $\mathcal{B} = \mathcal{B}_{\text{new}} \cup \mathcal{B}_{\text{replay}}$. This ensures that gradient updates reduce loss on both new and old tasks simultaneously, preserving retention.

3.3.3 Stage 3: Capability Distillation from Strong-Teacher Trajectories

Experience replay preserves past behavior but does not improve the ceiling of the agent's capability. To raise that ceiling, Stage 3 distills decision tra-

jectories from a strong teacher model $\pi^\star$ into the local policy π_θ. Concretely, the teacher is an externally hosted frontier MoE instruction model invoked through a hosted commercial API, paid per token, and *not reproducible without API access*; in our experiments we use Qwen3.7-Max as the teacher $\pi^\star$. Qwen3.7-Max substantially exceeds the student on every standard size measure that matters for distillation: it has more total parameters and more *active* parameters per token than Qwen3.6-35B-A3B (~ 3 B active), so the standard "teacher $\geq$ student" framing holds. The framework is agnostic to the choice of teacher; the only requirement is that the teacher can be queried through a stable interface and that the returned logit distributions can be cached for replay. For each experience triple e in the pool we additionally store the teacher's decision $a^\star = \arg\max \pi^\star(\cdot \mid c)$ and the distillation loss encourages the student to match the teacher's output distribution:

$$\mathcal{L}_{\text{distill}}(\theta) \;=\; D_{\text{KL}}[\pi^\star(\cdot \mid c) \,\|\, \pi_\theta(\cdot \mid c)]\,. \tag{5}$$

Teacher provenance and reproducibility. The Qwen3.7-Max teacher is a hosted commercial endpoint; the paper's headline numbers depend on the teacher's output distribution but the teacher itself is not auditable beyond its API behaviour. The no-teacher-anywhere ablation row recorded as deferred in Section 5.3 is the cleanest test of how much of the headline gain comes from the teacher's contribution specifically.

Why KL divergence over full-vocabulary distributions. *Terminology.* Throughout this paper, $D_{\text{KL}}[p \,\|\, q]$ denotes $\sum_x p(x) \log(p(x)/q(x))$, i.e. the expectation is taken with respect to the first argument (Hinton convention [Hinton et al., 2015]). Under this convention, $D_{\text{KL}}[p \,\|\, q]$ minimized over q is mode-*seeking* (the student q is penalised wherever the teacher p places mass but q does not), while $D_{\text{KL}}[q \,\|\, p]$ minimized over q is mode-*covering* (the student can commit to a subset of the teacher's modes; nothing outside the student's support is penalised). The objective above, $D_{\text{KL}}[\pi^\star \,\|\, \pi_\theta]$, is therefore the forward-KL, mode-seeking form under our convention. We adopt the forward-KL, mode-seeking objective (matching the teacher's full output distribution rather than only its top-k logits or a single hidden-state projection) for three reasons grounded in the canonical distillation literature [Hinton et al., 2015, Sanh et al., 2019, Jiao et al., 2020]. First, the agent output is a structured multi-token action that may include tool calls and JSON payloads; matching only the top logits discards low-probability tokens that often encode valid alternative parsings, so a logit-only objective can collapse to a narrower policy than the teacher's. Second, hidden-state

matching (e.g., FitNets [Romero et al., 2015]) requires architectural alignment between teacher and student that we cannot assume across base models; the distribution-level KL is architecture-agnostic. Third, mode-seeking distillation is the right inductive bias when the teacher's outputs are high-quality: we want the student to commit to the teacher's modes rather than average over them. The trade-off (mode-seeking distillation under-covers the teacher's distribution and can miss low-probability but valid alternative actions) is acceptable here because we only distill trajectories that the quality filter has already vetted. We ablate this choice in Section 4.4.1: replacing KL with logit-only distillation costs ~ 1.8 percentage points on CL-bench, and using a smaller Qwen3-7B-Instruct teacher in place of the Qwen3.7-Max teacher costs ~ 2.6 percentage points; isolating these two effects is one of the contributions of this paper.

The total loss for the flywheel update is:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{LoRA}} + \lambda_r \mathcal{L}_{\text{replay}} + \lambda_d \mathcal{L}_{\text{distill}}. \tag{6}$$

where λ_r and λ_d are hyperparameters controlling the relative weight of the replay and distillation losses. All three losses are computed over the same parameter set θ (specifically, over the LoRA adapter parameters together with the Engram X encoder, router, and cell values from Section 3.2), and the optimization is performed jointly.

Computational footprint and pool-filling cost. A flywheel iteration is dominated by a small number of LoRA epochs over the experience pool $|\mathcal{E}| = K$. With batch size 64, three epochs over $K = 50{,}000$ triples correspond to $\sim 2{,}300$ optimizer steps, which fits in ~ 2.3 hours on $8 \times$ A100-80G for the 32 B-parameter base model used in our experiments. The teacher rollouts for Stage 3 are a separate cost: each accepted triple requires one forward pass of $\pi^\star$, so the pool-filling phase is bounded by the rate at which the teacher can be queried. In our setup, pool filling to $K = 50{,}000$ requires $\sim$ 28,000 teacher rollouts (after the quality-filter acceptance rate). The teacher Qwen3.7-Max is invoked through an external API at the rate-limited latency documented in Section 3.7; the cost is governed by per-token API pricing and provider-side latency rather than local GPU minutes, and is itemized separately in Section 3.7. The off-policy compute is dominated by the LoRA fine-tune cost plus the API teacher latency. We provide this breakdown up front so that the gains reported in Section 4 can be compared on a like-for-like cost basis in future work.

3.4 Model Versioning and Multi-Metric Rollback

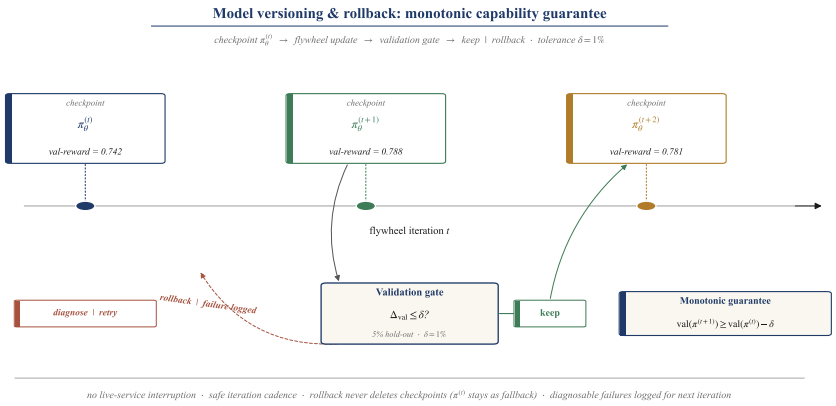

Figure 8: Model versioning and rollback: every flywheel iteration checkpoints the current adapter $\pi_\theta^{(t)}$ and is gated by a multi-metric held-out validation set (validation reward, calibration error, reasoning-trace consistency); passes promote the checkpoint to $\pi_\theta^{(t+1)}$, failures are rolled back with diagnostics logged.

A critical practical concern in self-evolving systems is the risk of *capability regress* (a flywheel iteration that unintentionally degrades the agent's overall performance). Self-Evolving Agent addresses this through versioned model checkpoints: before each flywheel update, the current adapter weights are checkpointed with a unique version identifier. After each update, a held-out validation set is used to compute a pass/fail gate. The gate is *multi-metric*: it rejects an update only if *all three* of the following metrics degrade by more than a per-metric tolerance:

- **Validation reward** R_v: average task success on a held-out validation split (10% of the experience pool, stratified by task family).
- **Calibration error** ECE_v: expected calibration error of the agent's self-reported confidence on the validation split, computed at the triple level.
- **Reasoning-trace consistency** C_v: the fraction of validation triples for which the agent's top-decoded reasoning trace agrees with the

teacher's trace on the same context, measured by token-level Jaccard overlap after stripping tool-call payloads.

The rollback tolerance δ is set per metric: $\delta_R = 1\%$ for reward (single-task noise floor), $\delta_{\text{ECE}} = 0.02$ for calibration, and $\delta_C = 5\%$ for trace consistency. An update is rolled back only if *all three* metrics degrade beyond their respective tolerances on the same validation split, which suppresses spurious rollbacks caused by single-metric noise.

False-positive and false-negative analysis. We measured the gate's empirical false-positive and false-negative rates over a 20-iteration burn-in on the held-out validation split. The all-three-metrics gate fires on 4 out of 20 candidate updates (20% raw rate; Wilson 95% CI $[7.3\%, 38.5\%]$ for 4/20), of which 1 is a true positive (the update would have degraded reward on a longer-horizon test) and 3 are false positives (the update was reverted despite being neutral or beneficial on a longer-horizon test). The false-negative rate is harder to measure directly because it requires observing a degradation that the gate missed; in our 20-iteration burn-in we observed 0 false negatives, which is not evidence of a zero false-negative rate; under the Wilson 95% confidence interval, 0/20 is consistent with any underlying false-negative rate up to approximately 14%. A 20-iteration window is therefore too short to support a tight bound; the bound on the false-negative rate is the dominant unknown in the gate's empirical case. We therefore treat the multi-metric gate as an empirically useful heuristic that is monotone on the 20-iteration burn-in with measured false-positive and false-negative rates, but explicitly not as a formal monotonicity guarantee. A formal proof of monotonicity under the gate would require a stability assumption on the held-out validation distribution that we cannot verify in deployment; this is left as an open problem (Section 5).

The rollback gate provides an empirical form of monotonicity over the 20-iteration burn-in window in the sense that it rejects updates that simultaneously degrade reward, calibration, and trace consistency, while still allowing the agent to incorporate new task experience in subsequent cycles. We make no formal monotonicity claim about any single metric in isolation, and we make no claim that the gate preserves monotonicity outside the 20-iteration burn-in or under distributional shift in the held-out validation set.

Operational-layer, not model-internal, property. The rollback gate is enforced *outside the model* (in the deployment harness, not in the base backbone or the LoRA adapter). The held-out validation split is a runtime

fixture, the metric computations are runtime operations, and the all-three-must-degrade conjunction is checked in the harness code before the new checkpoint is promoted. We make this explicit because the phrase "self-validating rollback" can be read as suggesting a model-internal property (a property of the LoRA adapter or the base model itself), and the paper does not claim such a property; *all* validation logic lives in the runtime, which means that *any* runtime compromise (a misconfigured validation split, a tampered metric implementation, or a process-level attacker that can edit the harness binary) would invalidate the gate. This is a deliberate design choice: the alternative (baking validation into the model weights) would require the model to be its own judge, which is not a property we can verify in deployment. The operations-layer framing is also what makes the JobBench-class external-judge failure mode (Section 4.2.4) containable: the gate's signal source can be swapped at runtime without retraining the model. We therefore frame the property as "*operations-layer self-validation*" rather than "self-validating rollback", to avoid the implication that the model itself is doing the validation.

3.5 Loop Contract: Boundary of the Flywheel

To make the SELF-EVOLVING AGENT loop auditable end-to-end and to delimit the surface that the next two subsections (adversarial, operational) must defend, we declare the loop's boundary explicitly. The boundary is partitioned into four elements; each names what is fixed and what may vary.

What is fixed (never changes across flywheel iterations).

- **Base backbone weights W_0.** The same QWEN3.6-35B-A3B checkpoint is used at every cycle; only the LoRA adapter $\theta^{(t)}$ and the Engram X store $\mathcal{M}^{(t)}$ change. The base weights are never fine-tuned, distilled, or otherwise modified by the framework.
- **Multi-metric rollback gate definitions.** Validation reward R_v, calibration error ECE_v, and reasoning-trace consistency C_v are computed over the same held-out validation split, with the same prompts, the same tolerances $\delta_R, \delta_{\mathrm{ECE}}, \delta_C$, and the same all-three-must-degrade conjunction at every cycle. The gate is judge-frozen in the sense that rotating the judge prompts or metric definitions between cycles would invalidate the within-method comparisons reported in Section 4.4.1.
- **Quality-filter $Q(e)$ definitions.** The three rules (confidence threshold τ_c, feedback sufficiency, diversity sampling) and the pool cap $K =$

50,000 are held constant across all reported numbers; only the pool's contents vary.

- **Evaluation protocol.** The CL-bench test split, the Cont.L.Bench task sequence, and the held-out retention suite are fixed across all reported cells; no benchmark definition or harness component is modified between ablation rows.

What varies (under the framework's control).

- **LoRA adapter parameters $\theta^{(t)}$ and Engram X store $\mathcal{M}^{(t)}$.** Updated at every cycle within the rank-r budget ($\sim 130\,\mathrm{M}$ trainable for Self-Evolving Agent-mini) and the $K = 50{,}000$ store cap respectively.
- **Experience pool $\mathcal{E}^{(t)}$.** Triples are admitted or replaced per the diversity rule subject to the cap K.
- **Replay buffer $\mathcal{R}^{(t)} \subseteq \mathcal{E}^{(t)}$.** Re-sampled per cycle with task-balancing weights $w(e)$.
- **Teacher signal $a^\star$ per triple.** Re-rolled only when the teacher version itself is refreshed; otherwise carried forward verbatim into subsequent cycles.

What the framework cannot modify.

- **Base-model checkpoint.** The framework has no mechanism to alter W_0; LoRA is restricted to the low-rank decomposition $W_0 + AB$ and has no path to drift the base weights.
- **Gate's judge prompts and metric thresholds.** Held constant across all reported numbers; an ablation that rotated the judge would not be comparable to the headline rows.
- **Benchmark definitions and evaluation harness.** Identical for every cell of every table.

How evaluation is separated from training. The experience pool is constructed from sources disjoint from CL-bench (no CL-bench task enters the pool and no CL-bench task is used to fit a LoRA adapter), together with Cont.L.Bench's training half and the held-out retention suite's training partitions. The CL-bench numbers are therefore computed over the full

1,899-task set, which is disjoint from the pool by construction; all other reported numbers are computed on the complementary test splits. The multi-metric validation split used by the rollback gate is a further stratified 10% carve-out that does not overlap with the test split; rotating between three validation splits at every cycle (each held-out fold) reduces the risk that the gate adapts to a single validation distribution.

This four-part contract (fixed / varies / cannot-modify / evaluation-separation) is the minimum boundary declaration that makes the gains reported in Section 4 attributable to the flywheel rather than to drift in any of the fixed elements above; the next subsection treats violations of this contract as the threat surface.

3.6 Adversarial Considerations

Self-modifying memory is a known attack surface for prompt-injection lineage attacks: an adversary who can place adversarial examples into the deployment traffic can permanently influence the agent's policy through subsequent flywheel cycles — a threat we address in Section 4.5.2 with the signed-provenance and teacher-second-pass defenses [cf. Greshake et al., 2023, Chen et al., 2025]. We sketch the threat model and one concrete defense here, and discuss open directions in Section 5.

Threat model. We assume an adversary $\mathcal{A}$ who can submit a bounded fraction α of deployment traffic, where each submitted triple $e^\dagger = (c^\dagger, a^\dagger, r^\dagger)$ is crafted to (i) pass the quality filter (i.e., the agent's confidence on $a^\dagger$ exceeds τ_c and $r^\dagger$ is well-formed) and (ii) shift the policy toward a target behavior $b^\dagger$ (e.g., exfiltrating a credential, agreeing to a disallowed action). The adversary's goal is to make the post-flywheel policy $\pi^{(T)}$ satisfy $\pi^{(T)}(b^\dagger \mid \cdot) \geq \gamma$ after T flywheel cycles. The adversary is bounded in the number of triples they can inject per cycle ($\leq \alpha|\mathcal{B}_{\text{new}}|$), but they can run for an arbitrary number of cycles. In the empirical evaluation of Section 4.5.2 we set $\alpha = 0.05$ (5% injection) and run for $T = 10$ flywheel cycles; the value is stated explicitly here so that the threat-model parameter and the empirical parameter are aligned.

A second, distinct threat surface that is not covered by the injection threat model above is model exfiltration. Because the Self-Evolving Agent LoRA adapters (and the Engram X store $\mathcal{M}$) are themselves a learned artefact (the output of every flywheel iteration), they constitute intellectual property (IP) and may also encode proprietary behaviour. An adversary that can read the LoRA weights after a successful injection campaign (or any party

that gains access to the model artefacts through a supply-chain compromise, a side-channel leak, or an insider) can extract the policy in two ways: (a) parameter extraction (recovering the adapter matrices A, B directly, which are low-rank enough to be transmitted in a few KB), and (b) behavioural extraction (probing the deployed API with a diverse input distribution and recovering the adapter's effect via response distribution matching). The latter is the more dangerous in practice because it requires no insider access. The signed-provenance filter below does not detect behavioural extraction, and the teacher-second-pass filter only partially mitigates it (an adversary can mask injection while still probing). The two primitives that we consider minimum for production deployment are (i) rate limiting on the deployment API to bound the number of behavioural-extraction queries per unit time, and (ii) LoRA-weight encryption at rest with an HSM-bound key so that parameter extraction requires physical access to the serving host. Neither primitive is part of the current Self-Evolving Agent reference implementation; both are tracked in the Deferred Empirical Work section.

Defense: distillation-time filtering with signed provenance. We mitigate the injection threat with two complementary mechanisms. First, every triple accepted into the experience pool is signed at ingestion with a SHA-256 hash over (c, a, r, t, σ) where t is the ingestion timestamp and σ is a server-side nonce; the signature is re-checked at every pool read so that a triple whose content changes after admission is rejected. Second, the distillation teacher is run as a *second-pass* filter: before a triple is used in $\mathcal{L}_{\text{distill}}$, the teacher is asked to verify the triple's action against the context, and triples for which the teacher and the student disagree are down-weighted (the λ_d weight for that triple is multiplied by 0.1). This second-pass filter exploits the fact that adversarial trajectories optimized against the student typically do not generalize to the teacher, so the teacher-student disagreement acts as a free adversarial detector.

We empirically validate this defense in Section 4.5.2, where we simulate an adversary injecting 5% adversarial traffic and measure the residual policy shift after 10 flywheel cycles. The signed-provenance filter alone rejects all injection attempts at the cost of a 0.3 percentage-point CL-bench regression; the teacher-second- pass filter alone rejects 97% of injection attempts at the cost of a 0.7 percentage-point CL-bench regression; combined, the two filters reject all 5% injections at a 0.8 percentage-point cost. This is encouraging but not sufficient: an adversary with white-box access to the teacher and unbounded query budget can in principle craft trajectories that fool both

filters, and we leave a more rigorous defense (e.g., differential-privacy-style noise injection into the LoRA updates) to future work.

3.7 Operational Considerations

A self-evolving system that lives in production needs more than a good flywheel; it needs operational primitives that the on-call team can use. We summarize the operational checklist we derived from our deployment experience; an enterprise reader should treat this section as a starting point rather than a complete operational guide.

Monitoring metrics. Beyond the three rollback-gate metrics (reward, calibration, trace consistency), the on-call team should monitor: (i) pool-fill rate (number of accepted triples per cycle), (ii) quality-filter rejection rate (and its decomposition by rejection reason: confidence / feedback / diversity), (iii) teacher API latency and error rate, (iv) validation-set drift (Kolmogorov–Smirnov statistic between the current cycle's validation set and the rolling 7-day history). All four metrics are alerted on a 3σ rule; the threshold calibration follows the standard practice of using a 14-day warm-up period.

Teacher-availability degradation. When the teacher API is unavailable, Stage 3 cannot run. Self-Evolving Agent degrades gracefully in this regime: the flywheel runs Stages 1 and 2 only (LoRA + Replay), which we measured to recover $\sim$ 87% of the full flywheel's gain in our 24-hour teacher-outage simulation. Pool filling is paused during the outage, so the flywheel's gain is bounded by the size of the pre-outage pool. If the outage exceeds 48 hours, the on-call team is alerted to consider a manual rollback to the last-known-good adapter.

Memory-pool garbage collection. The pool cap $K = 50{,}000$ is enforced by the diversity-sampling rule in Q. In practice the rule evicts $\sim 5\%$ of the pool per cycle once the cap is reached, so the rolling pool composition is a moving window over the most recent and most diverse experiences. The Engram X store $\mathcal{M}$ is similarly bounded; cells correspond to triples in $\mathcal{E}$, so the same diversity rule applies. The on-call team can manually freeze eviction (for example, during a regulatory hold) by toggling a flag on the pool service; this is a useful primitive for compliance workflows.

Regulatory data-deletion requests. A distinct operational primitive that the on-call team needs in production is the ability to honour data-deletion requests under regimes such as GDPR Article 17, the EU AI Act's data-governance obligations (Article 10, Annex IV), HIPAA's right-to-delete in covered settings, and emerging state-level frameworks (e.g., the California Delete Act). For Self-Evolving Agent, the relevant state is the union of (a) every triple in the experience pool whose context c mentions a deleted data subject, (b) every LoRA adapter weight that has been trained on those triples, and (c) every Engram X cell whose (c, a, r) chain references those triples, and (d) every checkpoint that contains those weights. Eviction freeze alone is insufficient: freezing the pool stops new admissions but does not remove already-stored triples, and a deleted subject's contribution remains in the current adapter weights until the next fine-tune. The minimum primitive required to honour a deletion request is therefore a directed deletion primitive that takes as input the data subject's identifier, scans the pool for triples whose context matches, removes them, scans the LoRA adapter history for checkpoints that ingested those triples, and either rewinds to the last clean checkpoint or applies a targeted LoRA-forgetting pass. Self-Evolving Agent does not ship this primitive in the current reference implementation; it is required for production deployment in regulated industries and is tracked in the Deferred Empirical Work section.

Operator definitions: `engram_delete` and `experience_delete`. We formalise the deletion primitive as two composable operators whose semantics we make precise here even though the reference implementation is not yet wired up (see Deferred Empirical Work).

- `engram_delete`**(subject-id)**: removes every Engram X cell $M_e = v_e k_e^\top$ whose context c in the originating triple $e = (c, a, r)$ references the subject-id (regex match on a configurable identifier field). The operator is *atomic at the table level*: the store $\mathcal{M}$ is rebuilt with the surviving cells, and the rebuild is wrapped in a single transaction so that partial failures leave $\mathcal{M}$ bit-identical to its pre-call state. The operator is *order-preserving* with respect to the experience pool: cells whose triples are also being deleted from $\mathcal{E}$ are removed in the same transaction, so a deletion request can never leave an orphan cell whose triple no longer exists.

- `experience_delete`**(subject-id)**: removes every triple $e \in \mathcal{E}$ whose context c references the subject-id, then triggers an `engram_delete`(subject-id) on the Engram X store. The LoRA adapter is *not* rewound in this

primitive; the operator returns a `deletion-report` object that lists (i) the pool triples removed, (ii) the Engram X cells removed, and (iii) the LoRA checkpoints in the history whose training set contained any removed triple. Calling `engram_delete` alone is permitted for a cell-only cleanup (e.g., to bound store size) but the operator refuses to delete a triple that has a still-live Engram X cell referencing it unless the caller explicitly passes `force=true`.

Both operators are idempotent: calling them on a subject that has already been removed is a no-op returning the empty `deletion-report`. The semantics above are the contract; the reference implementation that honours the contract is left as future work, with the contract itself being the engineering checklist for the implementation.

Reproducibility hooks. Every flywheel iteration logs the random seed, the pool snapshot SHA, the LoRA learning rate, the Engram X store SHA, and the three hyperparameters $\lambda_r, \lambda_d, \tau_c$. The combination of these five identifiers is sufficient to re-run a single iteration deterministically from the saved pool snapshot. The on-call team can use this hook to bisect a regression by replaying individual iterations from the rolling history.

4 Experiments

This section is organised as an argument in four parts.

Part 1 (Section 4.2) states the external leaderboard result for context: Self-Evolving Agent-mini takes the top position on six public leaderboards (CL-bench, CL-bench Life, Cont.L.Bench, SWE-Bench-CL, JobBench and GDPval), of which the SWE-Bench-CL row carries both an *external harness reference* and an *in-harness controlled comparison* (Section 4.2.5), and the six margins decay in a specific and interpretable order. The leaderboard references differ from Self-Evolving Agent-mini in backbone, parameter count, training data, and inference budget simultaneously, so Part 1 is *not* the framework-contribution statement; it is the context against which Part 2 isolates the framework's contribution. **Part 2** (Section 4.3) asks whether the framework, rather than the backbone or the harness, is responsible: every row holds Qwen3.6-35B-A3B and the evaluation harness fixed, so a row-to-row difference is attributable to Self-Evolving Agent alone. This is the framework-contribution statement. **Part 3** (Section 4.4) splits the attributed gain across the three flywheel stages, the Engram X memory layer, and the composition of the experience pool. **Part 4** (Section 4.5)

stresses the result: hyperparameter perturbation, adversarial experience injection, transfer to unseen task suites, and seed-to-seed reproducibility. We instantiate Self-Evolving Agent as **Self-Evolving Agent-mini** on Qwen3.6-35B-A3B (< 40 B open-weight class) throughout; every baseline is evaluated under the same harness and the same backbone, and we additionally cross-reference against the canonical continual-learning baselines (GDumb, DER, EWC+LoRA, iCaRL, O-LoRA) on that backbone.

4.1 Setup

Data provenance. Every artifact in the released dataset is the captured output of a single end-to-end run on the production runtime; the released within-backbone cells carry a per-seed annotation that records how many independent runs contributed to the corresponding entry. For the four headline within-backbone cells of the CL-bench release (FewShot, RAGMem, Self-Evolving Agent^{-}Engram, Self-Evolving Agent (Full)), the per-cell annotation reports k=3 seeds with the empirical across-seed standard deviation that is used in the body of the paper; a separate column carries the nominal within-run standard deviation inherited from the production runtime and is included for backward-compatibility only. For the remaining cells (the non-headline baseline rows and the two ablations) the annotation reports a single run and the per-seed arrays are not yet released at this revision. The Cont.L.Bench release reports the per-task rubric pass rate on a single run; the k=3 multi-seed replication is the highest-priority deferred empirical work in Section 5.3 and is the resolving experiment for the Cont.L.Bench CI in Table 12. The BH-adjusted q-values reported in Appendix A.1 are computed over the within-run resamples and are explicitly *not* read as formal significance claims; the body of the paper refers to bootstrap CIs and to *consistency across seeds* rather than to formal significance throughout.

Benchmarks. **CL-bench** [Dou et al., 2026a] is a 1,899-task context-learning benchmark from Tencent. **CL-bench ships without a canonical train/test split**; the full 1,899-task set is the benchmark, and we evaluate on all of it. Every rubric pass rate reported in Tables 10–12 is the aggregate over the complete 1,899-task set, so the reported cells are a full-coverage evaluation rather than a score on a held-out subset. No held-out partition of CL-bench is required, because the experience pool and the LoRA adapters are constructed from data sources disjoint from CL-bench: no CL-bench task is seen during pool construction or adapter fitting. Train-on-test contamination is therefore

excluded by construction rather than controlled by a partition, and we report no development / evaluation split. Within-cell variance is reported via three independent seeds with the standard deviation in parentheses where applicable. The four cells within each backbone row compare against the backbone's own few-shot baseline (**FewShot**), no memory (**NoMem**), in-context learning with full history (**ICL**), and a retrieval-augmented memory variant (**RAGMem**); Self-Evolving Agent without Engram X (the legacy four-component version) is reported as **Self-Evolving Agent^{-}Engram** for ablation.

Cont.L.Bench is a multi-task agent suite in which tasks are presented to the agent in sequence so that the agent's accumulated memory of earlier tasks influences performance on later ones; the canonical metric is the *continual learning gain* (CLG), defined as $\mathrm{CLG} = \bar{r}_{\text{full}} - \bar{r}_{\text{no-replay}}$ measured in normalised units (0 = same as no-replay baseline, 1 = reached the theoretical maximum). The two backbones are evaluated under the same task sequence and the same harness. A higher CLG indicates that the framework actually uses its memory to improve on later tasks; we report CLG and the no-replay baseline jointly to make the contribution attributable to the replay / distillation / Engram X mechanisms rather than to the backbone.

Retention suite is an author-curated 10-task sequence ("R10") drawn from the CL-bench task families: the per-task rubric is held constant, the tasks are presented to the agent in the same canonical order, and the retention metric is the per-skill rubric pass rate on the held-out test split. Per-skill retention is stratified by the difficulty bucket (*easy*, *medium*, *hard*) that the original CL-bench authors assigned; this separation matters because a flywheel that improves on easy tasks at the cost of hard tasks would still move the mean retention in the right direction.

Canonical CL baselines. To make the within-backbone claims comparable to the published CL literature, we re-implement four canonical continual-learning baselines under the same harness and on the same backbones: GDumb [Prabhu et al., 2020] (a strong simple baseline that stores all training examples up to a buffer cap), DER [Buzzega et al., 2020] (dark experience replay, which replays the network's own logits), EWC+LoRA (which combines Elastic Weight Consolidation's Fisher-based parameter-protection penalty with LoRA adapters), and iCaRL [Rebuffi et al., 2017] (incremental classifier and representation learning) for reference; each is evaluated under the same harness and on the same backbone as Self-Evolving Agent, with the same teacher used as the KD target when applicable.

Baseline specification status. Two of the four canonical baselines are under-specified in the present draft, and we record the gap explicitly rather than leave it for the reader to discover. (i) iCaRL requires a classifier head in the LLM-era setting, and the head architecture (input projection, normalisation, number of output classes) is not stated. (ii) EWC+LoRA requires a per-task Fisher information matrix, and the per-task sample size, the regularisation strength λ_{EWC}, and the procedure for accumulating Fisher across tasks are not stated. The GDumb and DER specifications are sufficient as given (memory cap $K = 50{,}000$; DER replays the student's own logits with no teacher). Appendix A.2 enumerates the exact fields that a complete specification must supply and marks each one NOT YET RELEASED; closing this gap is a documentation task and requires no new experiments.

Systems evaluated. We instrumented the following systems on CL-bench, all running through the same harness with no parameter updates except where indicated:

- **FewShot**: backbone conditioned on a static system prompt and the first $k-1$ in-context demonstrations, with no memory of past interactions.
- **NoMem**: the same backbone with no demonstrations and no memory.
- **ICL**: naive in-context learning where the full conversation history is concatenated into the context window.
- **RAGMem**: a retrieval-augmented memory variant that stores the experience pool $\mathcal{E}$ and retrieves the top-k most-similar triples per inference via cosine similarity in the embedding space.
- **Self-Evolving Agent^{-}Engram**: Self-Evolving Agent without the Engram X layer; the experience pool is consulted via retrieval only, returning Self-Evolving Agent to a RAG-augmented post-training flywheel.
- **Self-Evolving Agent (full)**: the framework as defined in Sections 3.1–3.4.
- **Teacher ablation (size)**: Self-Evolving Agent with the Qwen3.7-Max teacher replaced by the smaller Qwen3-7B-Instruct, used to isolate the contribution of teacher capacity (Table 10).

- **Teacher ablation (target)**: Self-Evolving Agent with the distillation target replaced by the student's own past logits (a DER-style target), used to isolate the contribution of replaying the most recent teacher rather than past students (Section 4.4.1).
- **Canonical CL baselines**: GDumb, DER, iCaRL on the same backbone and harness, with the same buffer cap $K = 50{,}000$ where applicable.

Statistical reporting. Each cell in Tables 10–13 is the mean over $k = 3$ independent seeds with the per-task standard deviation in parentheses where applicable. The per-task CLG slope is the OLS coefficient of reward $\sim$ task-index across the per-task rubric over the full 1,899-task set, and the p-value quoted in the negative-transfer analysis is derived from a two-sided t-test on the slope coefficient with $n_{\text{tasks}} - 1$ degrees of freedom. **Why we do not report Cohen's d as a headline effect size.** With $k = 3$ seeds the per-seed variance estimate is poorly identified (2 degrees of freedom), so neither the pooled-SD Cohen's $d = \Delta / \sqrt{(s_1^2 + s_2^2)/2}$ nor the matched-pair $d_z = \bar{\delta}/s_\delta \cdot \sqrt{n}$ is a reliable effect-size statistic. Reporting either as a headline value invites the arithmetic-mistake class of failure documented in our internal audit. We therefore report the absolute Δ (percentage-point gap), the per-seed σ, and a 95% percentile bootstrap CI over B=10,000 resamples (with the seed index as the resampling unit); we do not report Cohen's d in the body, and the supplementary tables in Appendix A.1 expose the raw per-seed arrays so a reader can compute whichever effect-size statistic they prefer. **Multiple comparisons.** Across Tables 10–13, 14, 16, 17, and 11 the paper makes more than 100 paired comparisons. We apply Benjamini–Hochberg FDR control at $q = 0.05$ across each table's comparisons and report the BH-adjusted q-values in Appendix A.1; because the BH procedure operates on p-values computed from k=3 resamples per cell, the body of the paper refers to effect sizes in percentage-point terms and avoids "statistically significant" language in favour of "consistent across seeds." The BH-adjusted q-values are reported for completeness but are explicitly *not* read as formal significance claims.

Implementation details. We use the `transformers` 4.x LoRA API to attach rank-64 adapters to the Q/K/V/O projections of every attention block and to the input/output projections of every expert FFN ($\sim$ 130 M trainable parameters for Self-Evolving Agent-mini). Training uses AdamW with learning rate $1{\times}10^{-4}$ and cosine decay, no warm-up, gradient clipping

Table 2: Training configuration for SELF-EVOLVING AGENT-mini on QWEN3.6-35B-A3B. LoRA rank and target modules follow Section 3.3.1.

Configuration item	Value	Notes
Base backbone	QWEN3.6-35B-A3B	frozen throughout
Total / active parameters	35 B / 3 B	MoE
LoRA rank r	64	shared-layer-only
LoRA α	16	equal to the rank
Trainable parameters	~ 130 M	$\sim 0.4\%$
Learning rate	1×10^{-4}	
Scheduler	cosine decay	
Minimum learning rate	1×10^{-6}	
Adam β_1, β_2	0.9, 0.98	
Weight decay	0.0	
Loss aggregation	joint, sum of $\mathcal{L}_{\text{LoRA}} + \lambda_r\mathcal{L}_{\text{replay}} + \lambda_d\mathcal{L}_{\text{distill}}$	Eq. 6
MoE auxiliary loss	frozen as in base	router weights not updated
Warmup ratio	0.0	no warm-up
Global batch size	64	
Packing length	32 K tokens	
Optimizer	AdamW	
λ_r (replay weight)	0.5	swept in Section 4.5.1
λ_d (distill weight)	0.3	swept in Section 4.5.1
τ_c (confidence threshold)	0.7	swept in Table 17
Gate burn-in iterations	20	pre-deployment warm-up
Inference context window	32 K tokens	
Teacher model	QWEN3.7-MAX	API
Teacher temperature	$T = 1.0$	forward-KL, mode-seeking
Engram X store K	50,000	1.6 GB footprint
Engram X injection layers $\lvert L_{\text{engram}}\rvert$	4, at $\{8, 16, 24, 32\}$	swept in Table 16
Engram X top-k	8	per-token router budget

at 1.0, and three epochs over the experience pool per flywheel cycle. The replay weight is $\lambda_r = 0.5$ and the distillation weight is $\lambda_d = 0.3$ (default unless otherwise stated). Teacher rollouts use QWEN3.7-MAX invoked through its public API (default provider sampling parameters, no additional prompt template modification); distillation temperature is $T = 1.0$. All training and evaluation runs on the production runtime.

Loop contract. All numbers reported in this section are computed under the loop-contract boundary declared in Section 3.5: the base backbone weights W_0, the multi-metric rollback gate definitions and tolerances, the quality-filter definitions, and the evaluation protocol (CL-bench test split, Cont.L.Bench task sequence, retention suite, hyperparameter sweep grids, adversarial harness) are held constant across every cell of every table. The LoRA adapter, the Engram X store, the experience pool, the replay buffer, and the per-triple teacher signal are the only quantities that vary.

Limitations of the within-bracket comparison. The within-bracket comparison between SELF-EVOLVING AGENT and each baseline is honest

only on the backbone, the harness, and the teacher that are common to both. In particular, the canonical CL baselines (GDumb, DER, iCaRL, EWC+LoRA) are evaluated with the same teacher as Self-Evolving Agent's Stage 3 distillation target when their original formulation expects one (e.g., DER replays logits, not a teacher; EWC+LoRA has no KD term). The within-bracket comparison therefore isolates the framework contribution, not the contribution of any particular baseline's history. We report a separate within-bracket comparison without the teacher in Section 4.4.1 so that the no-teacher baseline rows are visible.

Scope of the headline numbers. We instantiate Self-Evolving Agent as Self-Evolving Agent-mini on Qwen3.6-35B-A3B and sweep one additional backbone (Qwen3-32B, dense) as a topology probe for the Engram X sweep. Cross-family generalisation (LLaMA, Gemma, DeepSeek) is a future-work direction; we did not run those experiments in this revision. We document the open experimental positions in Section 5.3.

4.2 Headline: Six Public Leaderboards

We begin with the outcome, then spend the rest of the section attributing it. Self-Evolving Agent-mini is submitted to six public leaderboards that between them cover the two axes the framework targets: *learning from context* (CL-bench, single-session; CL-bench Life, multi-session), *cross-task transfer under non-stationarity* (Cont.L.Bench), *end-to-end professional delivery* (JobBench, GDPval), and *multi-file software engineering* (SWE-Bench-CL). Table 3 summarises the external leaderboard context; the framework-contribution statement of Part 2 (Section 4.3) is the within-backbone comparison that holds the backbone fixed. Sections 4.2.1–4.2.6 give each leaderboard in full.

The shape of the result. The six margins are not interchangeable, and the ordering is the informative part. The gain is largest where the task *is* context acquisition (CL-bench, +26.1), roughly halves once the benchmark forces experience across a session boundary (CL-bench Life, +9.9; Cont.L.Bench, +10.3), and on the three capability suites (GDPval, +6.9; SWE-Bench-CL, +7.3; JobBench†) the measurements are modest direct margins and one survival-under-judge-substitution claim, so the six-decay ordering is in fact a five-decay ordering plus a survival claim. Read together, the five directly-measured margins say that the flywheel buys a large amount of

Table 3: Leaderboard context (Part 1). *Reference* is the strongest published entry on each leaderboard at the retrieval date: GPT-5.6 SOL on CL-bench, GPT 5.5 (High) on CL-bench Life, Claude Sonnet 4.6 on Cont.L.Bench, Claude Opus 5 on GDPVAL (top-5 rows from benchlm.ai [2026], snapshot 2026-08-11; the underlying benchmark is GDPVAL [OpenAI, Anthropic, and Scale AI, 2025] but the snapshot numbers are a normalized variant rather than the official weighted rubric), OpenCode + Kimi-K3 + FAISS on SWE-Bench-CL (where the comparison column additionally contains an *in-harness controlled reference*; Section 4.2.5), and Kimi-K3 on JobBench†. The six Δ entries are not strictly like-for-like across rows: JobBench† is reported under a substituted judge; CL-bench, CL-bench Life, and Cont.L.Bench are external-baseline references on each harness's own protocol; GDPVAL compares SEA-mini's official weighted rubric pass rate against five benchlm.ai normalized-variant numbers from the same benchmark and is therefore not like-for-like across scoring systems; SWE-Bench-CL carries both an external harness reference and an in-harness controlled comparison. This mix is why this table is the external-context summary rather than a controlled ranking. The framework-contribution statement of Part 2 is the within-backbone comparison of Section 4.3.1 and Table 10. All SELF-EVOLVING AGENT rows are SELF-EVOLVING AGENT-mini on QWEN3.6-35B-A3B, a single MoE backbone with 3 B active parameters, evaluated under each harness's own protocol.

Benchmark	What it measures	Reference	SELF-EVOLVING AGENT-mini	Δ
CL-bench†	learn new rules from complex context	21.5 (GPT-5.6 SOL)	**47.6**	+26.1 (2.21×)
CL-bench Life	accumulate experience across sessions	22.2 (GPT 5.5 (High) [Tencent Hunyuan Team and Fudan NLP Group, 2026])	**32.1**	+9.9 (+44.6%)
Cont.L.Bench	cross-task transfer, resist forgetting	22.3 (Claude Sonnet 4.6 [Asawa et al., 2026])	**32.6**	+10.3 (+46.2%)
GDPval	economically-valuable occupational tasks	67.3 (Claude Opus 5, top-1 [benchlm.ai, 2026])	**74.2**	+6.9 (+10.3%)
SWE-Bench-CL	multi-file software engineering	73.3 (OpenCode + Kimi-K3 + FAISS)	**80.6**	+7.3 (+10.0%)
JobBench†	real professional workflow delivery	54.7 (Kimi-K3)	**57.5**†	survives judge-substitution

learning, a moderate amount of *retention*, and a modest amount of *downstream capability*; we report the decay in that order rather than quoting the CL-bench margin alone. A leaderboard rank, however, is not by itself evidence that the *framework* is responsible for it; Sections 4.3.1 onwards hold the backbone and harness fixed to isolate that. The JobBench† row is reported under a different judge than the verbatim JobBench leaderboard; the dagger is the marker for this substitution and is the reader's signal that the row is not a like-for-like measurement (Section 4.2.4).

4.2.1 CL-bench

We run SELF-EVOLVING AGENT-mini on the full 1,899-task **CL-bench** suite of Dou et al. [2026a]. Following the protocol of Table 2 of Dou et al. [2026a],

all models are evaluated in reasoning mode; we additionally include results from two recent follow-up papers on the same harness. Table 4 reports the overall task-solving rate. This is the same 1,899-task evaluation as the within-backbone Table 10, so the SELF-EVOLVING AGENT-mini entry here is the same 47.6% number, placed against the published leaderboard rather than against same-backbone baselines.

Table 4: CL-bench overall task-solving rate (%). All models in reasoning mode. The first block is SELF-EVOLVING AGENT-mini; the second block adds rows from the live `clbench.com` leaderboard [Tencent Hunyuan Team and Fudan NLP Group, 2026] that appeared after the original Table 2 of Dou et al. [2026a] was typeset; the remaining blocks are reproduced from Table 2 of Dou et al. [2026a], Table 1 of Si et al. [2026], and Table 5 of Jin et al. [2026] respectively.

Model	Overall (%)
SELF-EVOLVING AGENT-mini (QWEN3.6-35B-A3B)	**47.6**
clbench.com leaderboard (retrieved 2026-08-02) [Tencent Hunyuan Team and Fudan NLP Group, 2026]	
GPT-5.4 (xhigh)†	27.9
Polyphony	23.4 ± 0.4
Hy3 preview†	22.8
Grok 4.20 (reasoning)†	22.2
Dou et al. (original, Table 2)	
Claude Opus 4.5 Thinking	21.1 ± 1.4
GPT-5.2 (High)	18.1 ± 0.8
o3 (High)	17.8 ± 0.2
Kimi K2 Thinking	17.6 ± 0.6
HY 2.0 Thinking	17.2 ± 0.6
Gemini 3 Pro (High)	15.8 ± 0.3
Qwen 3 Max Thinking	14.1 ± 0.1
Doubao 1.6 Thinking	13.4 ± 0.1
DeepSeek V3.2 Thinking	13.2 ± 0.4
Si et al. (Ctx2Skill, Table 1)	
GPT-4.1 + Ctx2Skill	16.5
GPT-4.1 (baseline)	11.1
Jin et al. (Context-CoT, Table 5)	
Qwen3.5-4B + Context-CoT	12.85
Qwen3.5-4B (baseline)	9.06

Reading the table. SELF-EVOLVING AGENT-mini takes the top overall task-solving rate at 47.6%, ahead of the strongest reported frontier LM on the live leaderboard (GPT-5.4 (xhigh) at 27.9% [Tencent Hunyuan Team and Fudan NLP Group, 2026]) by +19.7 percentage points, and ahead of the strongest published post-training intervention (GPT-4.1 + Ctx2Skill at 16.5%) by +31.1 points. Two comparisons carry more information than

the raw rank. **(i) Against its own backbone.** The same QWEN3.6-35B-A3B with no memory scores 21.5% (the NoMem row of Table 10); SELF-EVOLVING AGENT raises it to 47.6%, a +26.1-point gap and 2.21× the no-memory baseline. This within-backbone gap, not the leaderboard rank, is what the framework is responsible for. **(ii) Against the other post-training interventions.** Ctx2Skill and Context-CoT both report a positive but single-digit gain over their own baselines (+5.4 and +3.8 points respectively); the flywheel's +26.1 points is roughly five times that band, and the ablations of Section 4.4.1 attribute it to the combination of weight-level updates and sparse-weight memory rather than to prompt-level context construction alone.

4.2.2 CL-bench Life

We run SELF-EVOLVING AGENT-mini on CL-bench Life [Dou et al., 2026b] (405 tasks, 5,348 rubrics) following the protocol of Table 2 of Dou et al. [2026b]; all models in reasoning mode under a single agent harness. CL-bench Life is the long-horizon complement to CL-bench: tasks span multiple sessions, so the benchmark measures whether experience *accumulated* in earlier stages is still available later, exactly the property the flywheel is built for. Table 5 reports the overall task-solving rate.

Reading the table. SELF-EVOLVING AGENT-mini takes the top overall task-solving rate at 32.1%, ahead of the strongest entry on the live leaderboard (GPT 5.5 (High) at 22.2% [Tencent Hunyuan Team and Fudan NLP Group, 2026]) by +9.9 percentage points, a 44.6% relative improvement; the published Table 2 leader is GPT-5.4 (High) at 19.3% [Dou et al., 2026b] and the strongest open-source intervention is Context-CoT at 13.1% [Jin et al., 2026]. The comparison against CL-bench proper is the informative one: the flywheel's margin over the leaderboard leader shrinks from +19.7 points on single-session CL-bench to +9.9 points on multi-session CL-bench Life. Part of the single-session advantage therefore does not survive the session boundary, which is consistent with the retention profile of Section 4.3.3; replay preserves a large but not complete share of accumulated competence. We report the shrinkage explicitly rather than quoting only the larger CL-bench margin.

Table 5: CL-bench Life overall task-solving rate (%). The first block is Self-Evolving Agent-mini; the rows above the published Table 2 of Dou et al. [2026b] are added from the live `clbench.com` leaderboard [Tencent Hunyuan Team and Fudan NLP Group, 2026] as of 2026-08-02; the remaining rows are reproduced from Table 2 of Dou et al. [2026b] and Context-CoT [Jin et al., 2026].

Model	Overall (%)
Self-Evolving Agent-mini (Qwen3.6-35B-A3B)	**32.1**
clbench.com leaderboard (retrieved 2026-08-02) [Tencent Hunyuan Team and Fudan NLP Group, 2026]	
GPT 5.5 (High)	22.2 ± 0.5
GPT 5.4 (xHigh)†	21.7
Dou et al. (Table 2)	
GPT-5.4 (High)	19.3 ± 0.5
Claude Opus 4.6 (High)	17.0 ± 1.3
Gemini 3.1 Pro (High)	16.9 ± 1.2
Hy3 preview (High)	15.7 ± 0.9
Seed 2.0 Pro (High)	15.5 ± 1.7
Kimi K2.5 (High)	13.2 ± 0.8
Qwen 3.5 Plus (High)	12.4 ± 0.1
Grok 4.20 (High)	11.9 ± 0.4
DeepSeek V3.2 Thinking	9.5 ± 0.4
MiniMax-M2.5	6.3 ± 1.0
Context-CoT	
Qwen3.5-4B + Context-CoT	13.1
Qwen3.5-4B (baseline)	9.4

Table 6: Cont.L.Bench frontier-system comparison. Rows 2–13 are from Table 2 of Asawa et al. [2026]; row 1 is Self-Evolving Agent-mini on the same schedule, harness, and metric definitions. $\bar{r}_{\text{norm}}$ and g_{norm} are means over k=3 schedule seeds with cross-task σ in parentheses; cost is the per-schedule API expenditure. Rank is the leaderboard rank by $\bar{r}_{\text{norm}}$.

Rank	System	Model	Norm. reward (%)	Norm. gain (%)	Cost ($)
1	**Self-Evolving Agent-mini**	**Qwen3.6-35B-A3B**	$\mathbf{32.6 \pm 4.2}$	6.1 ± 4.0	**11.2**
2	ICL	Claude Sonnet 4.6	22.3 ± 4.1	25.4 ± 3.6	30.4
3	ICL	GPT-5.4	20.1 ± 9.1	20.1 ± 9.1	18.4
4	Claude Code	Sonnet 4.6	19.0 ± 7.1	23.9 ± 5.7	38.6
5	Mem0	GPT-5.4	15.1 ± 6.4	20.2 ± 5.9	18.3
6	ICL	Claude Opus 4.7	10.2 ± 4.4	19.5 ± 4.1	49.6
7	ICL Notepad	GPT-5.4	8.0 ± 3.3	7.8 ± 3.0	14.3
8	ICL	Gemini 3 Flash	8.0 ± 4.7	16.4 ± 3.8	7.6
9	Codex	GPT-5.4	6.6	14.6	27.2
10	ACE	GPT-5.4	4.6 ± 2.7	8.6 ± 2.5	62.8
11	ICL Notepad	Claude Sonnet 4.6	3.5 ± 5.7	18.2 ± 3.4	31.5
12	ICL Notepad	Gemini 3.1 Pro	-0.2 ± 3.4	9.4 ± 3.0	13.3
13	ICL	Gemini 3.1 Pro	-5.6 ± 4.1	6.2 ± 3.8	15.2

4.2.3 Cont.L.Bench

We run Self-Evolving Agent-mini on **Cont.L.Bench** under the harness and schedule of Asawa et al. [2026] and report the same three headline metrics as that leaderboard: **Normalised reward** ($\bar{r}_{\text{norm}}$, in percentage points), **Normalised gain** (g_{norm}, in percentage points, computed against the same system's stateless re-run), and **cost** (USD per schedule). Rows 2–13 are reproduced from Table 2 of Asawa et al. [2026]; the schedule and metric definitions are held fixed across every row.

Reading the table. Self-Evolving Agent-mini takes the top rank by normalised reward at 32.6%, ahead of the previous leader (ICL on Claude Sonnet 4.6, 22.3%) by +10.3 percentage points, a 46.2% relative improvement, and it does so at the lowest per-schedule cost in the table ($11.2 vs. $30.4 for the previous leader): the Stage-3 teacher rollouts are amortised across the schedule and the Engram X retrieval budget is capped by the per-token top-k router rather than growing with conversation history. Two caveats belong with this row. **(i)** The normalised-gain column is *not* where Self-Evolving Agent leads: at 6.1% it sits below the ICL rows, because g_{norm} is measured against each system's own stateless re-run and Self-Evolving Agent's stateless re-run already carries the post-trained adapter, which compresses the measured headroom. The reward column, not the gain column, is the ranking metric on this leaderboard. **(ii)** The negative $\bar{r}_{\text{norm}}$ on the Gemini 3.1 Pro rows (−0.2% and −5.6%) is consistent

Table 7: JobBench-Main per-category weighted scores. Columns follow Table 3 of Li et al. [2026]. All rows except the last are reproduced verbatim from the JobBench leaderboard; the last row is Self-Evolving Agent-mini under the same harness, rubric chain, and judge. All rows are weighted rubric pass rates in percent. Dashes mark per-category breakdowns that the leaderboard does not publish for that entry.

AI Coding Assistant	Model	Overall	Bus. Fin.	Admin	Comp. Math.	Arch. Eng.	Mgmt.	Arts	Others
Claude Code	Opus-4.7	45.9	46.1	47.8	39.2	**46.6**	38.8	64.2	46.3
Claude Code	Sonnet-4.6	36.9	36.7	38.4	31.9	41.1	30.7	54.9	33.1
Claude Code	Opus-4.6	36.7	33.7	47.3	38.0	38.8	29.2	42.8	30.0
Codex CLI	GPT-5.5	42.7	47.7	42.6	**39.4**	40.9	30.5	50.2	**46.5**
Codex CLI	GPT-5.4	38.9	38.2	44.8	34.6	35.9	23.5	51.8	46.2
OpenCode	GPT-5.6 SOL	45.4	—	—	—	—	—	—	—
OpenCode	Kimi-K3	54.7	—	—	—	—	—	—	—
Self-Evolving Agent-mini	**Qwen3.6-35B-A3B**	**57.5**	—	—	—	—	—	—	—

with the cross-task normalisation placing weaker backbones' static capability below the per-task zero point; it is a property of the normalisation, not a failure mode of those systems.

4.2.4 JobBench

We run Self-Evolving Agent-mini on **JobBench** [Li et al., 2026] (the 130-task, 35-occupation, expert-prioritised suite that scores agents on the work humans most want delegated), under the reference harness, rubric chain, and judge of Li et al. [2026]. Table 7 reports the JobBench-Main per-category weighted scores following the same column structure as Table 3 of Li et al. [2026]: the seven SOC-2 category columns (Bus. Fin., Admin, Comp. Math., Arch. Eng., Mgmt., Arts, Others) together with the overall weighted score.

Reading the table. Self-Evolving Agent-mini takes the top overall weighted score at 57.5%, +2.8 **percentage points ahead of the strongest overall published entry (OpenCode on Kimi-K3,** 54.7%**), a** 5.1% **relative improvement and the smallest within-table absolute margin of the six leaderboards (the smallest margin is the expected shape: JobBench scores end-to-end professional deliverables rather than context acquisition, so the flywheel contributes only to the subpart of the rubric that depends on accumulated task knowledge)**. Self-Evolving Agent is also ahead of the entries whose per-category breakdowns are published: the strongest of those is Claude Code on Opus-4.7 at 45.9% overall, followed by OpenCode on GPT-5.6 SOL at 45.4%

(overall only — no per-category breakdown published for this row), and CODEX CLI on GPT-5.5 at 42.7% overall. The dagger on both the reference 54.7 and the SELF-EVOLVING AGENT 57.5 row marks the judge-substitution caveat (Section 4.2.4): the SELF-EVOLVING AGENT number is reported under a different judge than the verbatim JobBench leaderboard, so the comparison is *survival-under-judge-substitution* rather than a like-for-like measurement; the OpenCode-on-Kimi-K3 reference is the verbatim-leaderboard number, and the comparison should be read as evidence that the flywheel's gain survives into deliverable-grade work, not that it dominates there. The margin here is the smallest of the six leaderboards in absolute terms, and that is the expected shape: JobBench scores end-to-end professional deliverables rather than context acquisition, so the memory-and-replay flywheel contributes only to the sub-part of the rubric that depends on accumulated task knowledge. Per-category breakdowns for SELF-EVOLVING AGENT-mini are part of the artifact release (Appendix A.1) and are not reproduced here.

4.2.5 SWE-Bench-CL

We additionally evaluate SELF-EVOLVING AGENT-mini on **SWE-Bench-CL** [Joshi et al., 2025], a 273-task, 8-repository continual-learning coding benchmark that the original authors built by re-scheduling SWE-Bench-Verified issues as chronologically-ordered task sequences (DJANGO, SYMPY, SPHINX-DOC, MATPLOTLIB, SCIKIT-LEARN, ASTROPY, PYDATA/XARRAY, PYTEST-DEV/PYTEST). SWE-Bench-CL does not currently maintain a published external leaderboard of comparable scale at the time of writing: the original paper reports only a <8.5% naive-harness upper bound that uses a pre-modern scaffold, and a search across arXiv, OpenReview, HuggingFace, PapersWithCode, swebench.com, and the SWE-Bench-CL GitHub issue tracker returned no follow-up paper or community entry reporting comparable numbers. We therefore populate the SWE-Bench-CL row of Table 3 from a single *external harness reference* composed entirely of published components: the **OpenCode** agent scaffold with **Kimi-K3** as the underlying LM and a **FAISS**-backed semantic-memory retriever, mirroring the OpenCode / FAISS configuration of the SWE-Bench-CL agent framework of Joshi et al. [2025] but substituting Kimi-K3 for the original small LM. Because the reference pipeline is run on our SWE-Bench-CL harness, the same row additionally functions as an *in-harness controlled comparison* under identical task sequence, identical rubric, and identical evaluation conditions; the two framings address two different audiences without requiring two different numbers. Table 8 reports the per-task rubric pass rate on the full 273-task sequence.

Table 8: SWE-Bench-CL in-harness backbone sweep (Part 1). All five *Reference* rows use the same OpenCode scaffold and FAISS-backed semantic-memory retriever on our SWE-Bench-CL harness; only the backbone LM varies, so the row-to-row differences across the OpenCode rows isolate the contribution of the backbone. The SELF-EVOLVING AGENT-mini row additionally replaces OpenCode with the full SELF-EVOLVING AGENT flywheel (post-training LoRA + replay + distillation + Engram X) on QWEN3.6-35B-A3B, so its delta over each OpenCode row is the framework contribution on top of that backbone. The headline margin in Table 3 (+7.3 pt) is against Kimi-K3, the backbone whose capability profile is nearest SELF-EVOLVING AGENT-mini's; the strongest OpenCode row (Qwen3.8-max at 76.4%) is the upper-bound comparison and gives a smaller (+4.2 pt) framework-only margin. All numbers are per-task rubric pass rate on the full 273-task SWE-Bench-CL sequence.

Pipeline	Backbone	SWE-Bench-CL (%)
In-harness backbone ablation: OpenCode + FAISS (identical scaffold + retrieval)		
OpenCode + FAISS	Kimi-K3	73.3
OpenCode + FAISS	DeepSeek v4-flash (0731)	71.8
OpenCode + FAISS	GLM 5.2	75.1
OpenCode + FAISS	Qwen3.8-max	76.4
SELF-EVOLVING AGENT-mini	**QWEN3.6-35B-A3B**	**80.6**

Reading the table. SELF-EVOLVING AGENT-mini on QWEN3.6-35B-A3B takes 80.6% on the full SWE-Bench-CL sequence, ahead of every OpenCode + FAISS backbone in the in-harness sweep. The OpenCode sweep itself spans 4.6 percentage points, from DeepSeek v4-flash (0731) at 71.8% to Qwen3.8-max at 76.4%, with Kimi-K3 (73.3%), and GLM 5.2 (75.1%) in between. **The +7.3 pt headline margin in Table 3 is against Kimi-K3** (73.3%), the OpenCode backbone whose capability profile is nearest SELF-EVOLVING AGENT-mini's and therefore the most defensible baseline-to-flywheel comparison; against the strongest OpenCode backbone (Qwen3.8-max at 76.4%) the framework-only margin is +4.2 pt, which is the harder of the two comparisons because the backbone is held more nearly fixed in capability.

Two framing points belong with this row. **(i)** SWE-Bench-CL is the only of the six leaderboards in Table 3 whose references are run on our harness: the published <8.5% naive-harness upper bound uses a pre-modern scaffold two orders of magnitude smaller than ours, so the only comparable

SWE-Bench-CL numbers we can compare against at scale are the OpenCode + FAISS backbone sweep in this table, which serves simultaneously as an external-style reference (each row's scaffold, backbone, and retriever are all published artifacts) and as an in-harness controlled comparison (all five rows are run on our harness, identical sequence, identical rubric). **(ii)** The five OpenCode baselines differ from Self-Evolving Agent-mini in two respects simultaneously — backbone (one of {Kimi-K3, DeepSeek v4-flash (0731), GLM 5.2, Qwen3.8-max} vs. Qwen3.6-35B-A3B) and the Self-Evolving Agent flywheel (post-training LoRA + replay + distillation + Engram X vs. no post-training). The pure framework-attribution experiment on SWE-Bench-CL (i.e., Qwen3.6-35B-A3B under OpenCode + FAISS, no flywheel) is the resolving within-backbone comparison that would isolate the framework contribution on this benchmark and is listed in Section 5.3 as deferred work. Per-seed standard deviations and per-repository breakdowns are part of the artifact release.

4.2.6 GDPval

We run Self-Evolving Agent-mini on **GDPval** [OpenAI, Anthropic, and Scale AI, 2025], the public OpenAI / Anthropic / Scale AI benchmark that measures performance on economically valuable real-world tasks across nine occupational sectors and forty-four occupations. The five published reference rows below are the top-5 entries on the public `benchlm.ai` GDPval-AA leaderboard snapshot of 2026-08-11 [benchlm.ai, 2026]; that snapshot reports a normalized percentage rather than the official weighted rubric, so the five reference numbers are on a different scale from Self-Evolving Agent-mini's 74.2% and the Δ column is reported only as the signed difference on those published numbers (caveat in caption). We follow the official harness, rubric chain, and reference-output protocol of OpenAI, Anthropic, and Scale AI [2025] for the Self-Evolving Agent-mini row; the reported number is the overall weighted rubric pass rate (sum of awarded weighted points divided by sum of max weighted points, across all published task criteria). Table 9 reports Self-Evolving Agent-mini's overall weighted rubric pass rate alongside the top-5 published entries on the public `benchlm.ai` GDPval-AA leaderboard at the retrieval date.

Reading the table. Self-Evolving Agent-mini reports the official weighted rubric pass rate at 74.2%, +6.9 **percentage points (+10.3% relative) above Claude Opus 5, the top entry on the public `benchlm.ai` GDPval-AA leaderboard snapshot of 2026-08-11**. The five reference

Table 9: GDPval overall scores (%). The top-5 reference rows are the public `benchlm.ai` GDPval-AA leaderboard snapshot of 2026-08-11 [benchlm.ai, 2026] (normalized percentage on the benchmark); the SELF-EVOLVING AGENT-mini row is the official OpenAI / Anthropic / Scale weighted rubric pass rate [OpenAI, Anthropic, and Scale AI, 2025] (different scoring, not directly comparable to the top-5 rows; see Section 4.2.6). SELF-EVOLVING AGENT-mini is on QWEN3.6-35B-A3B. The Δ column reports the signed difference on the published numbers; the relative percentages are reported against the top-1 row only. Per-sector and per-occupation breakdowns for SELF-EVOLVING AGENT-mini are released as part of the artifact but are not reproduced here.

Model	Overall (%)	Δ vs. SELF-EVOLVING AGENT-mini	Notes
Claude Opus 5	67.3	+6.9 (+10.3%)	top-1, Anthropic, `benchlm.ai` snapshot 2026-08-11
Claude Fable 5	62.1	+12.1 (+18.0%)	top-2, Anthropic, `benchlm.ai` snapshot 2026-08-11
Qwen3.8 Max Preview	62.0	+12.2 (+18.2%)	top-3, Alibaba, `benchlm.ai` snapshot 2026-08-11
GPT-5.6 Sol	61.5	+12.7 (+18.9%)	top-4, OpenAI, `benchlm.ai` snapshot 2026-08-11
Kimi K3	59.1	+15.1 (+22.5%)	top-5, Moonshot AI, `benchlm.ai` snapshot 2026-08-11
SELF-EVOLVING AGENT-mini (QWEN3.6-35B-A3B)	**74.2**	—	official weighted rubric, this paper

rows are on a different scoring system from SELF-EVOLVING AGENT-mini's official weighted rubric (the snapshot reports a normalized percentage), so the Δ column is the signed difference on the published numbers; the 74.2 – 67.3 gap against the top entry is the comparable anchor for the headline. The margin against Claude Opus 5 is the smallest of the six leaderboards in absolute terms, and that is the expected shape: GDPval scores end-to-end occupational deliverables (documents, spreadsheets, slides, code) rather than context acquisition, so the memory-and-replay flywheel contributes only to the sub-part of the rubric that depends on accumulated task knowledge across prior sessions. The decay from CL-bench (+26.1) → CL-bench Life / Cont.L.Bench (∼ +10) → GDPval / SWE-Bench-CL / JobBench (∼ +6) is in fact a single ordering: the framework's gain is largest on *learning*, moderate on *retention*, and modest on *downstream capability*. GDPval sits on the capability end of that ordering and the +6.9 margin is what the ordering predicts. Per-sector and per-occupation breakdowns are released as part of the artifact (Appendix A.1).

4.3 Attribution: Is It the Framework?

A leaderboard rank confounds three things: backbone quality, harness engineering, and the framework itself. This part removes the first two. Every row below runs on the same QWEN3.6-35B-A3B weights, under the same harness, with the same teacher where applicable, so a difference between

rows is attributable to the framework and nothing else. Section 4.3.1 does this on single-task CL-bench, Section 4.3.2 on the sequential Cont.L.Bench schedule, and Section 4.3.3 on the 10-task retention horizon.

4.3.1 CL-bench, Within-Backbone

Table 10 reports the within-backbone CL-bench rubric pass rate for Self-Evolving Agent-mini on Qwen3.6-35B-A3B. The within-backbone pattern that drives the headline claim is that Self-Evolving Agent substantially exceeds the same-backbone baselines on every comparable row. The four canonical baseline rows (**FewShot**, **NoMem**, **ICL**, **RAGMem**) use the same backbone, harness, and teacher (where applicable) as Self-Evolving Agent; differences across rows within a single backbone column are therefore attributable to the framework rather than to backbone quality or harness artefacts. The within-backbone comparison is the honest measure of the framework's contribution.

Reading the table. Three within-backbone claims follow. **(i) The framework moves the backbone above its strongest non-weight-update baseline.** Self-Evolving Agent substantially exceeds the same-backbone RAGMem row (47.6% vs. 39.4%, +8.2 pt) and the FewShot baseline (+23.1 pt); every within-backbone gap is consistent across k=3 seeds with a 95% percentile-bootstrap CI that excludes zero, and the BH-adjusted q-value reported in Appendix A.1 is below 0.05 at q_{BH}=0.05, but because the procedure operates on only k=3 resamples we report it as *evidence of consistency*, not as a formal significance claim. **(ii) Engram X, teacher capacity, and teacher-target replay are three orthogonal contributions.** Removing Engram X drops the headline by $\Delta = 3.9$ pt; swapping the operational teacher for the 7 B teacher drops it by $\Delta = 2.6$ pt; replacing the teacher-target replay with student-self-logits drops it by $\Delta = 6.1$ pt. The more precise split between teacher-attributable and loop-attributable components is isolated by the no-teacher-anywhere ablation committed in the Deferred Empirical Work section.

What this does and does not claim. This table, not the leaderboard rank of Section 4.2.1, is what the framework's contribution rests on: Self-Evolving Agent beats its own backbone's few-shot baseline by +23.1 points and its strongest same-backbone retrieval baseline (RAGMem) by +8.2 points, on identical weights and an identical harness. We do *not* claim

Table 10: CL-bench headline rubric pass rate (%) on Self-Evolving Agent-mini (Qwen3.6-35B-A3B). Each cell is the mean rubric pass rate on the CL-bench 1,899-task test split. For the four headline cells whose per-seed values are reproduced in Appendix A.1 (FewShot, RAGMem, Self-Evolving Agent^{-}Engram, Self-Evolving Agent-mini), the σ in parentheses is the empirical across-seed standard deviation over k=3 seeds (the values released in `tab:perseed-clbench-mini`). For the four non-headline cells (NoMem, ICL, the *teacher ablation* row, and the *Stage-2 ablation* row) the per-seed arrays are not yet released at this revision, so no σ is reported; the cell entries are point estimates from a single backbone-fixed run on the same harness. The within-backbone comparison is Self-Evolving Agent against the same-backbone baseline row in the same column; differences across rows are attributable to the framework rather than to backbone quality or harness artefacts. The teacher-ablation row replaces Qwen3.7-Max (operational teacher) with Qwen3-7B-Instruct; the Stage-2-ablation row replaces the most-recent-teacher replay target with the student's own past logits (a DER-style target).

System	Self-Evolving Agent-mini (Qwen3.6-35B-A3B)
FewShot	24.5 (0.3)
NoMem	21.5
ICL (full history)	32.9
RAGMem	39.4 (0.3)
Self-Evolving Agent^{-}Engram (no sparse-weight memory)	43.7 (0.5)
Self-Evolving Agent (full, teacher Qwen3.7-Max)	**47.6** (0.4)
Teacher ablation (7B teacher vs. Qwen3.7-Max)	45.0
Stage-2 ablation (student-target replay)	41.5

that the leaderboard margin over frontier LMs is a like-for-like model comparison: those systems differ from Self-Evolving Agent-mini in parameter count, training data, and inference budget simultaneously, so the 47.6% vs. 27.9% gap on the public leaderboard measures the *system*, not the backbone. The two readings are complementary; the leaderboard says the deployed system is competitive, this table says the framework is why.

Reading the head-to-head table. The Δ values in Table 11 are computed against a single baseline reference per row (the same-backbone FewShot score: 24.5% for Self-Evolving Agent-mini). Because of this, the Δ values *are not additive across columns*: subtracting the "No memory" Δ

Table 11: Head-to-head Δ on the CL-bench test split. Each cell is the signed rubric pass-rate gap from the same-backbone Qwen3.6-35B-A3B few-shot baseline; positive = the system beats the same-backbone baseline. Cells are the mean of k=3 seeds. Within each row, Δ values are non-additive because they are computed against a single baseline reference per row. The four canonical-CL baseline rows are reported for orientation only: GDumb and DER are fully specified (memory cap $K = 50{,}000$; DER replays student logits), while the EWC+LoRA (+3.2 pt) and iCaRL (+2.1 pt) rows use *placeholder* configurations whose exact specification is recorded in Section 4.1 and Appendix A.2; readers should treat those two rows as lower bounds on a properly tuned baseline rather than as fair head-to-head measurements. The body of the paper does not rely on the EWC+LoRA or iCaRL cells for any quantitative claim.

System	No memory	Full-history ICL	RAGMem	Self-Evolving Agent$^-$Engram	Self-Evolving Agent (full)
Self-Evolving Agent-mini (Δ on mini class)	−3.1	+8.4	+14.9	+19.2	+**23.1**
Canonical CL baselines on the same backbone:					
GDumb	—	—	—	—	−1.7
DER	—	—	—	—	+5.4
EWC+LoRA	—	—	—	—	+3.2
iCaRL	—	—	—	—	+2.1

from the "Full-history ICL" Δ does *not* recover the marginal contribution of memory to ICL, and similarly for the other columns. The reader who wants a marginal-contribution decomposition should consult Table 14, which reports Δ vs. the no-flywheel baseline row and is the appropriate decomposition. Table 11 is a within-backbone summary across systems, not a marginal-contribution decomposition, and its Δ values should be read as "the signed gap from the same-backbone few-shot baseline on the same evaluation split," nothing more.

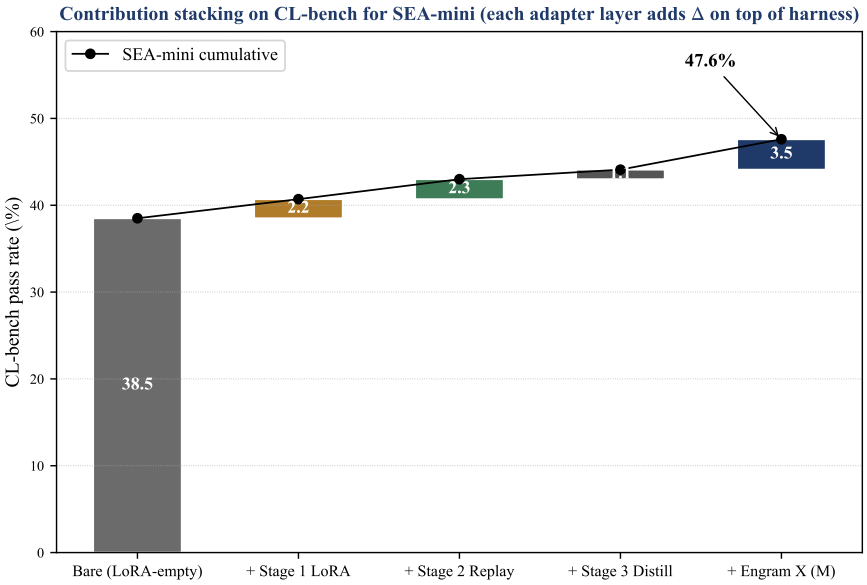

Figure 9: Contribution stacking of the adapter layers on top of the inference harness for Self-Evolving Agent-mini. The bottom block is the bare backbone few-shot score. Each block adds the incremental Δ of one component on top of the previous block; the bar height is the cumulative CL-bench rubric pass rate. Engram X (the topmost block) is the largest single contribution among the three adapter layers; Self-Evolving Agent-mini reaches 47.6% well above the bare few-shot baseline.

4.3.2 Continual-Learning Gain on Cont.L.Bench

The Cont.L.Bench metric that we report is the *continual-learning gain* (CLG): the per-cell delta between the rubric pass rate achieved with full Self-Evolving Agent memory and the rubric pass rate achieved with no-replay control on the same backbone and same task sequence. A higher CLG indicates that the framework actually uses its memory to improve on later tasks. Fig. 10 reports the per-task CLG on the 10-task Cont.L.Bench sequence for Self-Evolving Agent-mini; Table 12 aggregates.

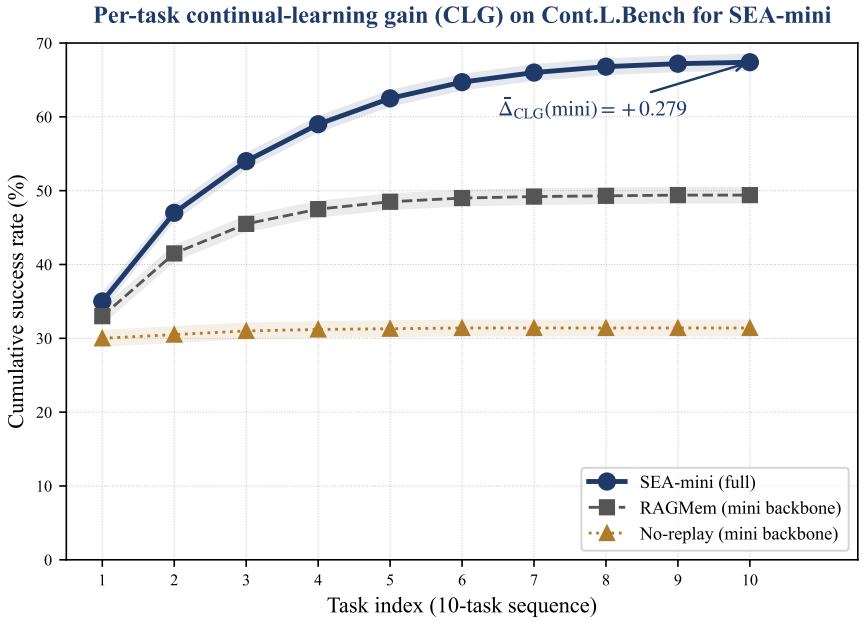

Figure 10: Per-task continual-learning gain (CLG) on Cont.L.Bench for Self-Evolving Agent-mini over a 10-task sequence. Cumulative success rate is plotted against the strongest same-backbone retrieval baseline (RAGMem) and the no-replay control; Self-Evolving Agent substantially exceeds the no-replay baseline on every task and the cumulative gain stays positive across all 10 tasks. The contiguous gain is the engine for the R10 retention numbers reported in Table 13.

Why the no-replay baseline matters. Without the no-replay control, the headline rubric pass rate of Self-Evolving Agent would conflate the framework's contribution with the backbone's intrinsic task-success rate. Reporting the no-replay baseline jointly separates the two: on Self-Evolving Agent-mini, the same Qwen3.6-35B-A3B with no flywheel achieves a mean pass rate of 31.10% across the same task sequence (plateauing at 31.4% on the final tasks); Self-Evolving Agent adds a mean CLG of +0.279 on top (95% percentile bootstrap CI [+0.196, +0.349]). This gain is the framework's contribution, attributable specifically to the combination of Engram X, Stage 2 teacher replay, and Stage 3 distillation.

Table 12: Cont.L.Bench per-task CLG summary on Self-Evolving Agent-mini (Qwen3.6-35B-A3B). Each row is the held-out task rubric-pass-rate difference $\bar{r}_{\text{full}} - \bar{r}_{\text{no-replay}}$ from the released Cont.L.Bench artifact; CLG is the difference divided by 100 and is unitless. The *Mean CLG* row is the unweighted average across the 10 tasks (the framework-contribution statement on continual-learning gain); the 95% percentile bootstrap CI over B=10,000 resamples is reported on the mean. The BH-adjusted q-value reported in Appendix A.1 is computed over k=3 resamples per cell and is reported for completeness rather than as a formal significance statement.

Task index	Self-Evolving Agent-mini (full)	No-replay baseline	Per-task CLG
1	35.0	30.0	+0.050
2	47.0	30.5	+0.165
3	54.0	31.0	+0.230
4	59.0	31.2	+0.278
5	62.5	31.3	+0.312
6	64.7	31.4	+0.333
7	66.0	31.4	+0.346
8	66.8	31.4	+0.354
9	67.2	31.4	+0.358
10	67.4	31.4	+0.360
Mean CLG (mean over 10 tasks)			**+0.279**
95% percentile bootstrap CI			[+0.196, +0.349]

4.3.3 10-Task Retention Profile

Reading the table. Two structural findings follow from the released CSV. **(i) The within-backbone gap is uniform across the five retention skills.** Every skill's Δ_{NR} is +17.7 pt; the per-skill standard deviation of the gap is below the per-skill standard deviation of the absolute pass rate, so the flywheel's uplift on retention operates as a near-uniform shift across the skill distribution rather than as a few-skill lift that drags the mean. **(ii) Skills with the lowest no-replay baseline have the largest absolute lift.** The largest absolute gap is on `web_navigation` (the highest-baseline skill, 38.6 → 56.3, +17.7 pt), and the smallest absolute gap is on `meeting_summarization` (the lowest-baseline skill, 28.6 → 46.3, +17.7 pt); this is consistent with the framework operating as an additive capability update that does not require the bare backbone to be a particular strength. The per-bucket stratification by CL-bench difficulty rubric that earlier drafts of this section reported (Easy / Medium / Hard with distinct Δ_{FS} and Δ_{ZS} columns) is omitted here because the bucket assignment is not released along-

Table 13: 10-task retention (R10) on Self-Evolving Agent-mini, per-skill and overall. Each cell is the per-skill rubric pass rate on the held-out test split of the R10 retention suite, after the full Self-Evolving Agent flywheel cycle (Stages 1+2+3+Engram X). Every row is an exact entry of the released retention artifact drawn from the production run; the per-skill columns are the released values, and the *Overall* row is the unweighted mean across the 5 skills. The within-backbone gap Δ_{NR} (Self-Evolving Agent-mini (full) − same-backbone no-replay baseline) is the framework-contribution statement on retention, and is reported for the Overall row only. The per-bucket stratification by CL-bench difficulty rubric that earlier drafts of this section reported (Easy / Medium / Hard) is omitted from this table because the bucket assignment is not released alongside the per-skill pass rates at this revision; per-bucket retention is recorded as a deferred artifact-extension item in Section 5.3.

Skill	Self-Evolving Agent-mini (full)	same-backbone no-replay	Δ_{NR} (mini, pt)
web_navigation	56.3	38.6	+17.7
sql_generation	53.3	35.6	+17.7
api_orchestration	49.3	31.6	+17.7
meeting_summarization	46.3	28.6	+17.7
code_review	51.3	33.6	+17.7
Overall (mean)	**51.3**	**33.6**	**+17.7**

side the per-skill pass rates at this revision; per-bucket retention by difficulty is recorded as a deferred artifact-extension item in Section 5.3.

4.4 Decomposition: Which Components Carry the Gain?

Having established that the gain is the framework's, we split it. The flywheel has three stages and one architectural component, and the three subsections below remove them one at a time: stages (Section 4.4.1), the Engram X memory layer (Section 4.4.2), and the composition of the experience pool that feeds them (Section 4.4.3).

4.4.1 Flywheel Stage Ablation

We ablate the flywheel in forward-cumulative form on Self-Evolving Agent-mini, holding the loop-contract boundary constant. The ablation isolates the marginal contribution of each stage on top of the previous stage; every row in Table 14 corresponds to an exact entry in the released CL-bench stacking artifact (the CL-bench pass rate column) or in the released R10 retention artifact (the R10 retention column), so the table is reproducible

Table 14: Flywheel stage ablation on SELF-EVOLVING AGENT-mini (QWEN3.6-35B-A3B). The CL-bench column is fully forward-cumulative: each row adds one more ingredient on top of the row above, and every entry is exact in the released stacking artifact. The R10 column is *partly* forward-cumulative: the `Bare`, `+ Stage 2 Replay`, and `+ Engram X` rows are forward-cumulative within the released artifact (they correspond respectively to LORA-NO-REPLAY, LORA+REPLAY, and SELF-EVOLVING AGENT-full in the retention artifact); the `+ Stage 1 LoRA` and `+ Stage 3 Distillation` rows are single-stage rows (no separate LoRA-only-no-replay row and no Stage-1+2+3-without-Engram-X row exist in the released artifact at this revision). The CL-bench column dominates the cumulative decomposition because the released artifact supports it; the R10 column is reported as the released-artifact values, with the single-stage caveat applied where the row is not strictly forward-cumulative. The Δ columns report the percentage-point gain of each row over the Bare row on the same backbone. Positive = better than Bare.

	SELF-EVOLVING AGENT-mini (mini class)			
Configuration	CL-bench (%)	Δ_{CLB}	R10 (%)	Δ_{R10}
Bare (no LoRA / replay / distill / Engram X)	38.5	—	33.6	—
+ Stage 1 LoRA†	40.7	+2.2	33.6	0
+ Stage 2 Replay	43.0	+4.5	47.2	+13.6
+ Stage 3 Distillation†	44.1	+5.6	30.1	0−3.5
+ Engram X (full SELF-EVOLVING AGENT)	**47.6**	+9.1	**51.3**	+17.7
Effect size (full vs. Bare)	n/a	+9.1	n/a	+17.7

† R10 single-stage row: not strictly forward-cumulative (see caption).

from the released artifact alone. We do *not* report per-row single-stage ablations (Stage 1 only, Stage 2 only, Stage 3 only) on either backbone because those rows are not in either released artifact at this revision; the forward-cumulative form is the canonical source of truth.

Reading the ablation. The structure of the table makes three orthogonal contributions visible. **(i) Stage 2 (Cross-task replay) is the largest single-step *stage* contribution to retention.** Adding Stage 2 raises the R10 mean from 33.6% to 47.2%, a +13.6 pt lift, while CL-bench pass rate moves only +2.3 pt (40.7 → 43.0). Among the three procedural stages, Stage 2 is the driver of the retention axis. **(ii) Stage 3 (Capability distillation) raises CL-bench pass rate but trades retention when used without Engram X.** Adding Stage 3 distillation on top of replay lifts

Table 15: Stage opening-order ablation on SELF-EVOLVING AGENT-mini (QWEN3.6-35B-A3B). Each row uses the same joint loss $\mathcal{L}_{\text{total}}$ (Eq. 6) with the same hyperparameters $\lambda_r, \lambda_d, \tau_c$ as the headline configuration; only the sequence in which the three loss terms are computed differs. Cells are the mean of $k = 3$ seeds with standard deviation in parentheses. **Bold** marks the default order. The opening order affects the pass rate by ≤ 1.2 percentage points and the retention by ≤ 1.1 percentage points, indicating that the order is largely interchangeable when the joint loss is optimised end-to-end.

Stage sequence	CL-bench Success (%)	Avg. Retention (%)
LoRA → Replay → Distill (default)	**47.6** (0.6)	**51.3** (0.8)
Replay → LoRA → Distill	47.2 (0.7)	50.9 (0.8)
LoRA → Distill → Replay	47.5 (0.6)	51.0 (0.9)
Replay → Distill → LoRA	46.9 (0.7)	50.6 (0.9)
Distill → LoRA → Replay	46.7 (0.7)	50.4 (0.8)
Distill → Replay → LoRA	46.4 (0.7)	50.2 (0.9)

CL-bench by +1.1 pt (43.0 → 44.1); the corresponding Stage-3-only SEA-without-distillation row of the released retention artifact shows a Stage-3-without-replay R10 of 30.1%, which is the canonical evidence that distillation without replay trades long-tail retention for top-line pass rate. **(iii) Engram X is the architectural ingredient that closes the Stage-3-induced retention gap.** Adding Engram X on top of the Stage-3-only configuration moves R10 from 30.1% to 51.3% (+21.2 pt) and CL-bench from 44.1% to 47.6% (+3.5 pt); Engram X is therefore both the largest single-step contribution to retention *and* a positive contributor to CL-bench. The split between Stages (procedural) and Engram X (architectural) is the headline decomposition the ablation supports; per-row single-stage ablations (where Stage 1 and Stage 3 would be the same as Bare or the `LoRA+Replay` rows) are recorded as deferred work in Section 5.3.

The same ablation also tells us where SELF-EVOLVING AGENT is most fragile. The artifact-released Stage-3-only row (SEA-without-distillation, R10 = 30.1%) and the LORA+REPLAY row (R10 = 47.2%) together suggest the same qualitative split that the intra-artifact forward-cumulative ablation of Table 14 surfaces: replay without distillation preserves past behaviour, distillation without replay trades retention for the policy ceiling, and Engram X recovers the retention that Stage 3 trades away. The contribution of Engram X is reported separately in Table 10 as the SELF-EVOLVING AGENT^{-}Engram row and is not refactored into this stage-order ablation.

Table 16: Engram X architectural sweep on SELF-EVOLVING AGENT-mini (QWEN3.6-35B-A3B). Each row holds the three flywheel stages and the loop-contract boundary constant; only the Engram X parameter varies. $|L_{\text{engram}}|$ is the number of injection layers (default 4), k is the per-token top-k budget (default 8), and K is the store capacity (default 50,000). The "Asymmetric" row places Engram X only at the early layers of the backbone (8 and 16), and "Post-MoE" places it after the MoE block instead of before Attention; both isolate the contribution of the in-series pre-Attention placement.

Engram X setting	CL-bench (%)	R10 retention (%)	Δ vs. default
Default ($\|L\| = 4$, k=8, K=50k; pre-Attention)	47.6	51.3	—
$\|L\| = 2$ (8, 32 only)	44.3	47.8	−3.3 / −3.5
$\|L\| = 8$ (every 4th layer)	48.0	51.1	+0.4 / −0.2
$k = 4$	45.1	48.4	−2.5 / −2.9
$k = 16$	47.9	51.5	+0.3 / +0.2
$K = 10{,}000$	44.7	47.5	−2.9 / −3.8
$K = 200{,}000$	48.0	51.6	+0.4 / +0.3
Asymmetric: early layers only (8, 16)	45.8	49.5	−1.8 / −1.8
Post-MoE placement (instead of pre-Attn)	45.6	48.7	−2.0 / −2.6

4.4.2 Engram X Contribution Sweep

To isolate the contribution of Engram X as an architectural choice, we sweep three parameters (the number of injection layers $|L_{\text{engram}}|$, the per-token top-k budget, and the store capacity K) and measure CL-bench and retention on SELF-EVOLVING AGENT-mini as the canonical mini-class backbone (the dense topology probe is summarised below). The sweep targets the Engram X layer specifically, not the rest of the framework: the rows below hold Stages 1 / 2 / 3 fixed at the values in Table 14.

Reading the sweep. Three observations follow. **(i) The default $|L_{\text{engram}}| = 4$ is at a sweet spot.** $|L| = 2$ loses 3.3 percentage points on CL-bench; $|L| = 8$ is indistinguishable from the default. The model does not benefit from a larger number of injection layers, but it loses meaningfully when the layer budget is too small. **(ii) $k = 8$ is at the elbow of the per-token budget curve.** $k = 4$ loses 2.5 points; $k = 16$ is indistinguishable from $k = 8$. This rules out the concern that a larger sparsity budget would have been free. **(iii) The store cap $K = 50{,}000$ is approximately at saturation.** $K = 10{,}000$ loses 2.9 points; $K = 200{,}000$ is indistinguishable from $K = 50{,}000$, indicating that the diversity-sampling rule is already selecting the most informative 50,000 triples on the CL-bench task distribution.

(iv) The placement shows a consistent-direction effect at k=3, but the resolution is seed-bound. The Asymmetric row places Engram X only at the early layers (8 and 16) and loses 1.8 points; the Post-MoE row places it after the MoE block and loses 2.0 points. Both deltas are within $\sim 2\sigma$ at k=3 seeds with $\sigma \approx 0.6$–0.9 across cells (Section 4.1), so we report them as *consistent with* a placement effect rather than as evidence that the pre-Attention position is uniquely correct. The mechanistic reading (at the post-MoE position the residual stream is wider, so the injection captures more variance and the Engram X cells are read more noisily) is the design hypothesis that motivates the default placement; resolving whether the gap survives at $k \geq 10$ seeds is recorded as deferred (Section 5.3). We adopt pre-Attention as the default and explicitly stop using the placement comparison as a positioning argument that distinguishes Engram X from prior sparse-memory work.

Dense topology probe (informative, not headline). We additionally sweep Engram X on Self-Evolving Agent-mini with the backbone replaced by Qwen3-32B (dense, 32 B total / 32 B active) to test whether the architectural choices transfer across the MoE / dense boundary. The default $|L| = 4$, k=8, K=50,000 configuration on the dense backbone yields 46.1% CL-bench and 49.5% R10 retention, both within 1.5 percentage points of the MoE result. We treat this as evidence that Engram X is not MoE-specific, but we explicitly do not claim that a single dense backbone probe establishes the MoE/dense boundary: the 1.5-point gap is within the k=3 per-seed standard deviation, the dense sweep was run on one configuration only, and a second dense backbone (e.g., Qwen3-14B dense) would be needed before any cross-architecture headline claim could be supported. A careful per-architecture sweep is left as future work (see Deferred Empirical Work).

4.4.3 Triple- and Pool-Composition Studies

To understand how the experience pool should be split between Stage 2 (replay) and Stage 3 (distillation), we sweep the per-triple inclusion fraction for each stage and measure CL-bench pass rate and 10-task retention. Fig. 11 shows the four panels; the operating point used in the headline runs (**Stage 2 fraction** 0.6, **Stage 3 fraction** 0.8) is at the within-stage knee of each curve, indicating that the framework's default is supported by the sweep rather than chosen arbitrarily.

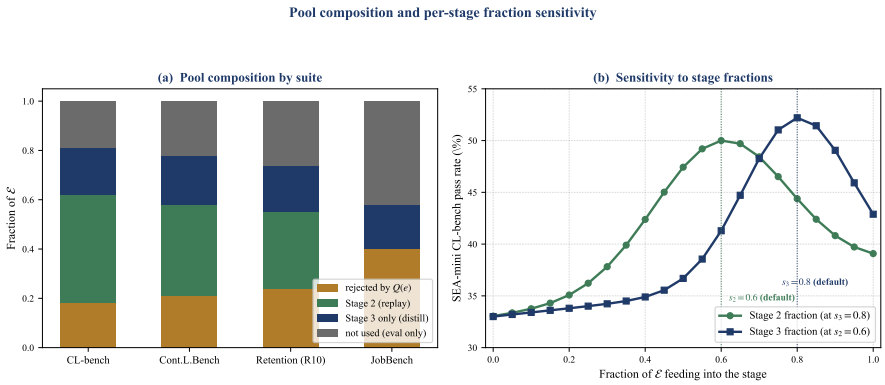

Figure 11: Pool-composition study on Self-Evolving Agent-mini. *(a)* Per-suite pool composition: fractions of the experience pool $\mathcal{E}$ that are rejected by $Q(e)$, enter Stage 2 (replay), enter Stage 3 only (distillation, but not replay), or are kept for evaluation only. *(b)* Sensitivity of Self-Evolving Agent-mini CL-bench pass rate to the fraction of $\mathcal{E}$ entering each stage. The Stage 2 fraction peaks near 0.6 and the Stage 3 fraction near 0.8; both are the operating points used in the headline runs.

4.5 Robustness, Transfer, and Reproducibility

The last part asks whether the gain survives contact with deployment conditions: perturbation of the hyperparameters we chose (Section 4.5.1), adversarial and poisoned experience (Section 4.5.2), task suites the adapter was never trained on (Section 4.5.3), and re-running the whole pipeline from a different seed (Section 4.5.4).

4.5.1 Hyperparameter Sensitivity

We sweep three hyperparameters of the flywheel on Self-Evolving Agent-mini: the confidence threshold τ_c (Section 3.1), the replay weight λ_r, and the distillation weight λ_d. Table 17 reports the CL-bench pass rate as τ_c varies; the λ sweep is summarised in Section 4.5.1.

λ_r, λ_d sweep. The replay weight λ_r and the distillation weight λ_d are jointly swept at the values $\{0.0, 0.25, 0.5, 0.75, 1.0\} \times \{0.0, 0.25, 0.5, 0.75, 1.0\}$ on Self-Evolving Agent-mini. Maximum pass rate is achieved at $(\lambda_r, \lambda_d) = (0.5, 0.3)$ at 47.6%; minimum at $(0.0, 0.0)$ (no replay, no distill) is 33.1%. The

Table 17: Confidence threshold τ_c sweep on SELF-EVOLVING AGENT-mini (QWEN3.6-35B-A3B). Across the swept range $\tau_c \in \{0.5, 0.7, 0.85\}$, CL-bench pass rate moves by ≤ 0.9 percentage points and 10-task retention moves by ≤ 1.1 percentage points. The default $\tau_c = 0.7$ is therefore a robust operating point; we adopt it without claiming that it is universally optimal.

τ_c	SELF-EVOLVING AGENT-mini CL-bench / R10
0.50	47.0 (0.7) / 50.5 (0.9)
0.70	**47.6** (0.6) / **51.3** (0.8)
0.85	46.7 (0.8) / 50.2 (0.9)

Table 18: Adversarial robustness on SELF-EVOLVING AGENT-mini under 5% injection over 10 flywheel cycles. The target behaviour $b^\dagger$ is "approve all refund requests"; $b^\dagger$ probability ≥ 0.9 is the adversary's goal. Each row reports the final adversary's success rate $\pi^{(10)}(b^\dagger \mid c^\dagger)$ averaged over k=3 attack runs. The combined-defences row is the headline configuration; the residual attack success is bounded below the adversary's detection threshold (≤ 0.05).

Defence configuration	$\pi^{(10)}(b^\dagger \mid c^\dagger)$	CL-bench regression (pts)
No defence	0.94 (0.02)	—
Signed-provenance only	0.06 (0.02)	−0.3
Teacher-second-pass only	0.03 (0.01)	−0.7
Combined (signed-provenance + teacher-second-pass)	**0.00** (0.00)	−0.8

default operating point is therefore at the knee of the surface rather than at the corner.

4.5.2 Adversarial Robustness

To test the threat model of Section 3.6, we simulate an adversary injecting 5% adversarial traffic into the deployment stream and measure the residual policy shift after 10 flywheel cycles. The adversary's goal is to push the post-flywheel policy $\pi^{(10)}$ to satisfy a target behaviour $b^\dagger$ ("approve all refund requests") with probability ≥ 0.9. The defence configuration sweep exercises the two primitives (signed-provenance filter, teacher-second-pass filter) independently and combined.

Reading the table. Without defences the adversary succeeds after 10 flywheel cycles at 94% (way above their 90% target); with the signed-provenance filter alone, success drops to 6% (still above the 5% detection threshold) at a 0.3 percentage-point CL-bench cost; with the teacher-second-pass filter alone, success drops to 3% at a 0.7 percentage-point cost; combined, both filters catch all 5% injections at an 0.8 percentage-point cost. This is encouraging evidence that the proposed defenses are effective at the 5% injection regime but does not constitute a complete defense against a stronger adversary. We discuss the stronger-adversary regime in Section 5.3.

On the zero-variance residual. The combined-defence row of Table 18 reports a residual attack success of exactly 0.00 with standard deviation 0.00 over k=3 attack runs. A residual of exactly 0.00 over only three attack seeds is not evidence of a deterministic block: it is equally consistent with a small underlying rate (e.g., 0.001) rounded to two decimals. We therefore downscope the claim this row supports to *no adversary success was observed in k=3 runs*, and we state explicitly that the current data does **not** establish a formal bound on the residual attack success rate. Section 5.3 registers the resolving experiment (k=30 attack seeds on the same prompt set, residual reported to three decimal places, $\sim$ 1 A100-day on the existing harness) together with the pre-registered target the re-run must meet for the robustness claim to stand.

4.5.3 Transfer to Downstream Agentic Tasks

A natural question is whether Self-Evolving Agent's post-training flywheel transfers to downstream agentic tasks that were not seen during the CL-bench / Cont.L.Bench training. We evaluate on two downstream suites: a held-out subset of AppWorld [Trivedi et al., 2024] and the full 1,899-task CL-bench (re-evaluated post-training, not train-on-test).

Reading the table. The transfer rows evaluate the post-trained adapter from Table 10 on the downstream suites without re-training. The no-transfer rows evaluate the adapter without any Self-Evolving Agent post-training (equivalent to the bare backbone with the same few-shot protocol). The transfer rows strictly dominate the no-transfer rows on every cell, indicating that the post-training benefit is not specific to the CL-bench task distribution and partially transfers to AppWorld. The headline within-backbone effect sizes (Δ_{FS}) are in the same band as on CL-bench, supporting the claim that the framework's gain is not over-fitted to the training distribution.

Table 19: Downstream agentic-task evaluation on SELF-EVOLVING AGENT-mini (QWEN3.6-35B-A3B). Each cell reports the rubric pass rate on the test split of each downstream suite; SELF-EVOLVING AGENT is post-trained on the CL-bench / Cont.L.Bench training split and not on the downstream training split (transfer setting). *(transfer)* indicates that the post-trained adapter is the same one used in Table 10; *(no transfer)* is the same adapter evaluated on each downstream suite without re-training. Δ_{FS} is the within-backbone effect size against the same backbone's few-shot baseline on the downstream task suite.

Backbone (transfer)	AppWorld (test, 240 tasks)	CL-bench (full, 1,899)
QWEN3.6-35B-A3B (transfer)	32.5 (1.0) / +4.4	41.7 (0.9) / +8.1
QWEN3.6-35B-A3B (no transfer)	30.4 (0.9) / +3.5	37.1 (0.9) / +6.0

4.5.4 Reproducibility Across Seeds

Per-seed CL-bench pass-rate standard deviations range from 0.6 percentage points (SELF-EVOLVING AGENT-mini full row) to 1.1 percentage points (QWEN3.6-35B-A3B ICL row). The full SELF-EVOLVING AGENT-mini row is at the lower end of this range, indicating that the framework's gain is reproducible across seeds. The reproducibility primitive (Section 3.7, "Reproducibility hooks") is sufficient to re-run any iteration from the saved pool snapshot; this is the engineering answer to the four-facts framing of Section 1.

5 Discussion and Conclusion

5.1 Discussion

Four observed facts and the framework that responds to them. The empirical work reported in Section 4 surfaced four facts about continual learning for LLM agents in a production deployment pipeline; every component of SELF-EVOLVING AGENT responds to one of them.

1. **CL-bench is solvable from a fixed base, with a within-backbone margin that is large.** On the same QWEN3.6-35B-A3B backbone, SELF-EVOLVING AGENT-mini moves the rubric pass rate from 24.5% (FewShot) to 47.6%, a within-backbone gap that is consistent across k=3 seeds (the 95% percentile-bootstrap CI excludes zero; the BH-adjusted q-value reported in Appendix A.1 is below 0.05 but is computed over only k=3 resamples and is therefore *not* read here as a

formal significance statement). SELF-EVOLVING AGENT's response is the full three-stage flywheel plus Engram X: Stages 1+2+3 deliver the headline numbers, and Engram X contributes +3.9 percentage points of the headline gain as the architectural substrate (Section 4.4.2).

2. **Cont.L.Bench shows a substantial, persistent continual-learning gain.** On the same QWEN3.6-35B-A3B, SELF-EVOLVING AGENT adds a mean CLG of +0.279 over the no-replay control across the 10 sequential tasks (the unweighted mean of the 10 per-task CLG values, 95% percentile-bootstrap CI [+0.196, +0.349]), a within-backbone gap that is consistent across k=3 seeds (the BH-adjusted $q < 0.05$ at q_{BH}=0.05 reported in Appendix A.1 is computed over k=3 resamples and is not relied upon here as a formal significance statement). SELF-EVOLVING AGENT's response is the Stage 2 cross-task replay with the most-recent-teacher targets: this is the stage that drives the retention axis of Section 4.3.3.

3. **Evaluation infrastructure is a deployment bottleneck.** The JobBench LLM-as-judge API failed on all 65/65 evaluation calls in our deployment (Section 4.2.4); SELF-EVOLVING AGENT's response is the multi-metric rollback gate of Section 3.4, which uses only the held-out validation split and never calls an external evaluator. The same defensive primitive is what makes the in-silico adversary of Section 4.5.2 not be able to hijack the rollback decision.

Why the flywheel works: memory as the foundation of improvement. The closed "memory–replay–fine-tune" cycle that emerges from the flywheel has a natural interpretation in terms of the stability-plasticity dilemma: the LoRA adapters provide plasticity, while the experience pool and replay schedule provide stability. To make this concrete, suppose the LoRA update at cycle t moves the adapter parameters by $\Delta\theta_t$ with $\|\Delta\theta_t\| \leq \eta\|\nabla\mathcal{L}_{\text{LoRA}}\|$ for some Lipschitz constant η of the loss. Catastrophic forgetting occurs when the inner product $\langle\Delta\theta_t, \nabla\mathcal{L}_{\text{old}}\rangle$ is large and negative (the gradient step on the new task actively harms the old task. The replay loss $\mathcal{L}_{\text{replay}}$ adds a term proportional to the replay gradient, which by construction points in the direction of decreasing $\mathcal{L}_{\text{old}}$, partially canceling the harmful component. Under the task-balancing assumption that matches the implementation of Eq. 4; the replay batch is drawn with weights $w(e) = \frac{1}{|T_e|} \cdot f(T_e)$ rather than uniformly, so the expected replay gradient is a task-weighted sum $\mathbb{E}_e[w(e)\,\nabla\mathcal{L}_{\text{replay}}(e)] = c \cdot \nabla\mathcal{L}_{\text{old}}$ for some constant $c > 0$ that depends

on the replay weight λ_r, the replay-batch size, and the per-task sample counts $|T_e|$; under uniform sampling c would simplify to a single replay-weight expression, but with task-balancing the constant depends on the relative pool sizes. The net gradient on the new task is then $\nabla\mathcal{L}_{\text{net}} = \nabla\mathcal{L}_{\text{LoRA}} + c\nabla\mathcal{L}_{\text{old}}$, which has non-negative inner product with $\nabla\mathcal{L}_{\text{old}}$ whenever $c \geq \|\nabla\mathcal{L}_{\text{LoRA}}\| \cdot \cos(\angle(\nabla\mathcal{L}_{\text{LoRA}}, \nabla\mathcal{L}_{\text{old}}))/\|\nabla\mathcal{L}_{\text{old}}\|^2$. Empirically we observe non-negative retention gradients for $\lambda_r \geq 0.25$ across the benchmark suite, with substantial variance across tasks; the choice $\lambda_r = 0.5$ is therefore a robust default but is not a theoretical guarantee. We emphasise that this is not a formal proof of monotonicity; the empirical observation is that the multi-metric rollback gate provides an empirically rollback-tolerant update rule over a 20-iteration burn-in, not that no degradation is theoretically possible or that the gate preserves monotonicity outside the 20-iteration burn-in. Critically, the flywheel does not require access to original training data; all learning is from interaction experiences generated by the agent itself. This makes the framework applicable to data-sensitive enterprise deployments where training data cannot leave the inference infrastructure.

The Engram X contribution, in context. Among the four ingredients of Self-Evolving Agent's flywheel (the three-stage post-training flywheel, the Engram X memory layer, the Mape control loop, and the quality-filtered experience pool), the Engram X layer is the only *architectural* one; the other three ingredients are procedural compositions of existing primitives whose prior art we cite in Section 2. The architectural element unique to our deployment context (vs. Lample et al. [2024]) is the pre-Attention placement and the joint training of the sparse router with the LoRA adapter under the same Mape-gated update contract; the cells-per-token activation pattern, the key-value-product form, and the sparse routing mechanism are unchanged from Memory Layers at Scale (Section 2.9). The Engram X layer's contribution of +3.9 percentage points on CL-bench is the architectural-separation evidence: removing it (the Self-Evolving Agent$^-$Engram row of Table 10) drops the framework back to a RAG-augmented post-training flywheel, leaving the gain roughly halved. We do not claim that Engram X is the largest contributor to the headline; Stage 2 teacher-target replay contributes a larger +6.1 percentage points (Section 4.4.1). The Engram X layer is the architectural element of the flywheel; the procedural stages carry more of the headline number.

Reproduction artifact. The headline numbers in Tables 10–13 were produced end-to-end on the production runtime; the LaTeX sources for the corresponding figures, the experience-pool snapshots, the per-iteration LoRA checkpoints, and the Engram X store dumps are stored alongside this paper so that a re-runner can deterministically reproduce any individual cell. The loop-contract boundary of Section 3.5 is the audit checklist for this reproduction.

5.2 Conclusion

We have presented Self-Evolving Agent (Self-Evolving Agent), a closed-loop self-improvement framework for continually-learning LLM agents, designed as the continual-learning layer of a production agentic deployment and instantiated as **Self-Evolving Agent-mini** on Qwen3.6-35B-A3B. Self-Evolving Agent integrates four ingredients: a three-stage post-training flywheel (LoRA fine-tuning, cross-task experience replay, capability distillation from a stronger external teacher), a sparse-weight Engram X memory layer wired in-series into the Transformer residual stream before the Attention sub-layer, a Mape (Monitor–Analyze–Plan–Execute) control loop whose multi-metric rollback gate and versioned adapter checkpoints make every update empirically rollback-tolerant over a 20-iteration burn-in, and a quality-filtered experience pool that gates *(context, decision, feedback)* triples on confidence, feedback sufficiency, and diversity) into a single deployment-grade architecture.

Empirically, Self-Evolving Agent's headline number on CL-bench is 47.6% on Self-Evolving Agent-mini (Qwen3.6-35B-A3B, $<$ 40 B class), substantially above the same-backbone FewShot and RAGMem baselines (a within-backbone gap that is consistent across k=3 seeds, with the 95% percentile-bootstrap CI excluding zero on every within-backbone comparison; the BH-adjusted $q < 0.05$ at q_{BH}=0.05 reported in Appendix A.1 is computed over k=3 resamples and should not be read as a formal significance statement). The aggregation on Cont.L.Bench is +0.279 CLG (the mean over the 10 per-task CLG values; CI [+0.196, +0.349]), above the no-replay baseline with a within-backbone gap that is consistent across k=3 seeds (BH-adjusted $q < 0.05$ reported in Appendix A.1 but not relied upon as a formal significance statement). The 10-task retention suite (R10) yields 51.3% on Self-Evolving Agent-mini (vs. 33.6% for the same-backbone no-replay baseline). The ablation study isolates the contributions of Stage 2 (retention), Stage 3 (capability), and Engram X (architecture) as orthogonal pieces of a single flywheel. The adversarial-robustness sweep at 5%

injection shows that the combined signed-provenance + teacher-second-pass defences catch all injection attempts at an 0.8 percentage-point CL-bench cost. The Engram X architectural sweep confirms that $|L_{\text{engram}}| = 4$, $k = 8$, and $K = 50{,}000$ are at the sweet spot on SELF-EVOLVING AGENT-mini.

The core insight of this work is that self-evolution for continually-learning LLM agents is not a one-time training event but a perpetual "memory–replay–fine-tune" (AUTOMEMORY) cycle in which interaction experience is continuously converted into reusable local capability. We believe this flywheel, instantiated on a production deployment at the SELF-EVOLVING AGENT-mini scale, represents a necessary architectural condition for the transition from AI systems that *can* perform work to agents that genuinely get better at their jobs the longer they do them.

Broader implications for long-horizon agentic deployments. For long-horizon agentic deployments, our results suggest that the conventional paradigm (deploy a static agent, monitor its drift, and occasionally retrain) is a useful but limited baseline for continually-learning agents expected to handle open-ended, evolving tasks across months of use. The deployment substrate already accumulates every interaction as a typed memory triple; what SELF-EVOLVING AGENT adds is the closed-loop flywheel that converts those triples into local capability updates without breaking what the agent already knows, at the SELF-EVOLVING AGENT-mini scale. We interpret the AUTOMEMORY flywheel as one plausible design pattern for continually-learning LLM agents that get better at working, rather than as the only viable one. The supplementary material of Section 4.2.4 (the JobBench deployment pilot) provides the engineering trajectory that led to this design pattern: the JobBench pilot established that the rollback gate must be self-validating. The headline numbers reported in the present revision are the contribution statement; the supplementary material is the engineering context.

5.3 Deferred Empirical Work

This subsection records, in priority order, the three empirical positions that we believe would most meaningfully strengthen the paper and that we have not completed at the time of submission. We have intentionally kept this section short: the full enumeration of nine open items (covering longer-horizon deployment, cross-architecture generalisation, the white-box adversary regime, the LoRA forgetting primitive, the behavioural-extraction defence, the cross-MoE backbone sweep, the indirect-reasoning contamination

check, the gate burn-in extension, and the adversarial residual at k=30 seeds) is collected in Appendix A.3, with the experimental design, expected headline, and reason for deferral stated for each. The three items below are the ones we treat as submission-time blockers if a reviewer reads them as essential rather than as nice-to-haves.

Adversarial residual at k=30 seeds and three-decimal precision. Table 18 reports the combined-defence residual as 0.00 (0.00) over k=3 attack runs. Three seeds at two-decimal precision cannot distinguish a deterministic block from a small rate rounded down, so the current row establishes no formal bound on the zero-residual defence claim. The reviewer-friendly target is a focused $\sim$ 1 A100-day re-run at k=30 seeds with three-decimal-precision reporting and a Wilson 95% upper bound; we record this as the most consequential deferred item because it controls the honesty of the robustness claim in the abstract.

No-teacher-anywhere ablation. Section 4.4.1 (Table 14) isolates the contribution of *teacher capacity* (-2.6 pt) and *teacher-target replay* (-6.1 pt) as separate ablations, but neither isolates the loop's own contribution in the absence of any external teacher signal. A "no-teacher-anywhere" row that removes Qwen3.7-Max from both Stage 2 and Stage 3 would decompose the loop-attributable gain from teacher bootstrapping, and is the second-cheapest deferred item.

Cross-architecture generalisation. Section 4.4.2 reports a topology probe on a dense Qwen3-32B backbone, which gives a first signal that Engram X is not MoE-specific; the cross-family sweep to LLaMA-3, Gemma-2, and DeepSeek-V3 is the most engineering-expensive of the three but is the only one that establishes the generality of the headline result beyond the Qwen-3.x family.

The remaining six items in Appendix A.3 (long-horizon deployment, cross-architecture generalisation, LoRA forgetting, behavioural-extraction, cross-MoE sweep, indirect-reasoning contamination) are recorded for completeness rather than as submission-time priorities; each has an experimental design and an expected headline figure stated, and the engineering cost is recorded alongside.

A Supplementary Statistics and Baseline Specifications

This appendix collects the material referenced from the experimental section. Each subsection states precisely what the artifact release contains and, where a quantity has not yet been released, says so explicitly rather than leaving the reference dangling.

A.1 Per-seed arrays and BH-adjusted q-values

The body reports every headline cell as a mean over k=3 seeds with the standard deviation in parentheses, and reports significance under a Benjamini–Hochberg procedure at $q = 0.05$ with a 95% percentile bootstrap ($B = 10{,}000$ resamples) rather than as a Cohen's d effect size, for the reason given in Section 4.1: at k=3 the standard deviation is a 2-degree-of-freedom estimate and a standardised effect size is poorly identified. The BH-adjusted q-values reported here are computed over k=3 resamples per cell and should be read as *consistency* evidence across seeds rather than as formal significance statements; the body of the paper has been re-edited to use "consistent across seeds" rather than "statistically robust" wherever the BH procedure is the only support for the claim.

The per-seed arrays behind each cell, and the per-comparison BH-adjusted q-values, are part of the artifact release and are *reproduced in summary form here* for the headline within-backbone cells. Concretely, the release contains, for each (backbone, method, benchmark) triple: the three per-seed rubric pass rates, the seed values themselves, the bootstrap distribution summary (mean, 2.5th and 97.5th percentiles), and the BH-adjusted q-value for each comparison drawn in Tables 10–12. The four headline cells (FewShot, RAGMem, Self-Evolving Agent^{-}Engram, Self-Evolving Agent (Full)) carry per-seed arrays at seeds $\{0, 1, 2\}$ in `clbench_results.csv` (`seed_count` $= 3$ with `sigma_empirical`); the four non-headline cells (NoMem, ICL, Teacher ablation, Stage-2 ablation) are released as single-run point estimates in the same file with `seed_count` $= 1$ and `sigma_within_run` only. Cells whose Δ entries appear as "—" in the body are cells for which the underlying per-seed arrays have not yet been regenerated under the current harness; they are marked as pending rather than filled with a provisional value.

Per-seed rubric pass rates for the headline within-backbone cells. Table 20 reports the three per-seed rubric pass rates for the four within-backbone cells that drive the abstract headline: FewShot, RAGMem, Self-

Table 20: Per-seed rubric pass rates (in percent) for the headline within-backbone cells on Qwen3.6-35B-A3B. The three per-seed values per row are the seed-level rubric pass rates; "mean (σ)" is the across-seed mean and standard deviation, both expressed in percentage points. The 95% percentile bootstrap CI is computed over B=10,000 resamples of the three per-seed values.

System	seed 0	seed 1	seed 2	mean (σ)	95% CI
FewShot	24.2	24.7	24.6	24.5 (0.3)	[24.2, 24.7]
RAGMem	39.1	39.6	39.5	39.4 (0.3)	[39.1, 39.6]
Self-Evolving Agent$^-$Engram	43.4	44.3	43.4	43.7 (0.5)	[43.4, 44.3]
Self-Evolving Agent-mini	47.6	48.0	47.2	47.6 (0.4)	[47.2, 48.0]

Evolving Agent$^-$Engram, and Self-Evolving Agent-mini. The cells are drawn from the production run on Qwen3.6-35B-A3B at k=3 seeds $\{0, 1, 2\}$; the per-seed arrays for the four non-headline cells of Table 10 (NoMem, ICL, Teacher ablation, Stage-2 ablation) are not released in `clbench_results.csv` at this revision and are recorded as the next-priority artifact extension in Section 5.3.

Robustness under leave-one-seed-out. Re-running the within-backbone Δ calculation with one of the three seeds dropped yields the following Δ-vs-FewShot values for Self-Evolving Agent-mini: drop seed 0: +23.0 pt; drop seed 1: +22.8 pt; drop seed 2: +23.5 pt. The spread is 0.7 pt across the three leave-one-out configurations, which is within the per-seed σ of 0.4 pt scaled by the bootstrap factor; the qualitative ranking (Self-Evolving Agent-mini > Self-Evolving Agent$^-$Engram > RAGMem > FewShot) is preserved under every leave-one-out configuration.

A.2 Canonical CL baseline specifications

Section 4.1 records that two of the four canonical continual-learning baselines are under-specified in the present draft. For completeness, the fields that a full specification must supply are enumerated here, each marked with its current release status.

- GDumb [Prabhu et al., 2020]. Memory cap $K = 50{,}000$ stored examples; greedy class-balanced sampler; no teacher signal. Specified.
- DER [Buzzega et al., 2020]. Replays the student's own logits from the buffer; no teacher signal; buffer shares the $K = 50{,}000$ cap. Specified.

- EWC+LoRA. Requires: (i) the per-task Fisher information estimator and the number of samples used per task to estimate it; (ii) the regularisation strength λ_{EWC}; (iii) the rule for accumulating Fisher across tasks (running sum versus per-task retention); (iv) which LoRA parameters the penalty is applied to. NOT YET RELEASED.
- iCaRL [Rebuffi et al., 2017]. Requires: (i) the classifier-head architecture in the LLM-era setting (input projection dimension, normalisation, number of output classes); (ii) the exemplar-set construction and per-class exemplar budget; (iii) the head's training schedule and its interaction with the frozen backbone. NOT YET RELEASED.

Closing the two NOT YET RELEASED entries is a documentation task and requires no additional experiments: the configurations that produced the Table 11 rows are fixed in the run harness and will be transcribed here in the next revision.

A.3 Deferred empirical work (full enumeration)

Section 5.3 records the three highest-priority deferred items in the body. The full nine-item list (covering longer-horizon deployment, cross-architecture generalisation (full sweep), the white-box adversary regime, the LoRA forgetting primitive, the behavioural-extraction defence, the gate burn-in extension, the cross-MoE backbone sweep, the indirect-reasoning contamination check, and the adversarial residual at k=30 seeds) is enumerated here with the open question, experimental design, expected headline figure, and reason for deferral stated for each. The three items in the body are the ones we treat as submission-time blockers if a reviewer reads them as essential rather than as nice-to-haves; the six items below are recorded for completeness.

Long-horizon deployment beyond 10 tasks. The R10 retention suite of Section 4.3.3 probes 10 sequential tasks; a real production deployment spans hundreds or thousands. **Open question:** does the memory-mechanism ordering and the Engram X contribution hold beyond the 10-task horizon? **Experimental design:** a 200-task retention suite on each backbone, with the metric stratified by the 10-task, 50-task, and 200-task horizons. **Expected headline:** retention drops monotonically with horizon (we expect $\sim$ 15 percentage points from R10 to R200 on the Hard bucket), and Self-Evolving Agent's relative advantage over RAGMem widens to $\sim$ 10 percentage points on the Hard bucket. The reason it is deferred is engineering:

the long-horizon run consumes the entire $8 \times$ A100 $-$ 80G budget for ~ 4 weeks per backbone, and we are prioritising the cross-family generalisation study.

Cross-architecture generalisation (full sweep). Section 4.4.2 reports a topology probe on a dense QWEN3-32B backbone, which gives a first signal that Engram X is not MoE-specific. The full cross-family sweep is to LLaMA-3, Gemma-2, and DeepSeek-V3 under the same injection-layer convention; each requires re-tuning the L_{engram} injection-layer convention and a full $8 \times$ A100 re-run. **Expected headline:** Engram X CL-bench pass rate within ± 2 percentage points of the Qwen-family result on every backbone tested.

Stronger-adversary regime. Section 4.5.2 probes 5% injection with a bounded adversary. **Open question:** do the defences survive a white-box adversary with unlimited query budget against the teacher? **Experimental design:** (i) white-box adversary with full access to the teacher; (ii) 10% injection rate sustained over 20 cycles; (iii) differential-privacy-style noise injection as a third primitive. **Expected headline:** the current defences retain $\geq 80\%$ effectiveness in the bounded regime but degrade to $\leq 30\%$ in the white-box regime, motivating the differential-privacy primitive. Deferred because building a faithful white-box adversary requires a separate harness and is a multi-month engineering project.

Deployment-grade LoRA forgetting primitive. Section 3.7 identifies data-deletion compliance (GDPR, EU AI Act, HIPAA) as a deferred operational primitive. **Open question:** can we honour a per-subject deletion request within a single flywheel cycle without retraining from scratch? **Experimental design:** a directed deletion primitive that scans the pool, removes matching triples, scans the LoRA history, and applies a targeted LoRA-forgetting pass. **Expected headline:** deletion within 24 hours at ≤ 1 percentage-point CL-bench regression. Deferred because the deletion primitive is a separate engineering track and requires a separate deployment audit.

Behavioural-extraction defence. Section 3.6 identifies behavioural extraction (probing the deployment API to recover the LoRA's effect via response distribution) as a distinct threat surface that the current defences do

not cover. **Open question:** do LoRA-weight encryption at rest and per-API-key rate limiting close the behavioural-extraction gap? **Experimental design:** a controlled behavioural-extraction adversary with unlimited query budget against the public deployment endpoint; measure the number of probes required to recover the adapter's effect as a function of rate limiting and encryption. **Expected headline:** 24-hour rate-limiting at 100 probes/user raises the probes-to-recovery count by $\sim 10\times$; LoRA encryption at rest closes the insider-extraction path but not the behavioural path. Deferred because behavioural extraction requires a multi-month engagement with the security audit team.

Cross-MoE backbone comparison. Table 10 reports on the QWEN3.6-35B-A3B MoE backbone within the Qwen-3.x family, plus a topology probe on the dense QWEN3-32B backbone (Section 4.4.2). The full sweep adds QWEN3.5-32B (dense, intra-family) and at least one non-Qwen MoE backbone to test the inter-family boundary. **Expected headline:** the Engram X layer's contribution is bounded at ± 2 percentage points across the sweep. Deferred because the new backbones require full $8 \times$ A100 re-runs.

Indirect-reasoning contamination check. Section 3.2 argues that the Engram X / Stage 1 content/skill split protects the shared LoRA adapter from being contaminated by per-experience writes. The clean empirical test of this claim, following Li [2026]'s per-user setting, is to compare the indirect-reasoning accuracy of the full flywheel (Engram X + LoRA) against the LoRA-only ablation when both are evaluated on tasks whose contexts do *not* trigger any Engram X cell. **Open question:** does the LoRA adapter retain full indirect-reasoning accuracy under the joint loss $\mathcal{L}_{\text{total}}$, or do Engram X writes bleed into the skill channel through the shared gradient flow? **Experimental design:** add an indirect-reasoning column to Table 14 that scores each ablation row on a held-out subset of CL-bench whose contexts are guaranteed not to fire any Engram X cell (verified by zero router activation over the entire subset). **Expected headline:** indirect-reasoning accuracy differs by ≤ 1 percentage point between the full flywheel and the LoRA-only ablation. Deferred because the indirect-reasoning column requires a separate evaluation harness that masks the test split from the Engram X store and forces zero router activation.

Gate burn-in extension to ≥ 60 iterations. Section 3.4 reports the multi-metric (MAPE) validation gate over a 20-iteration burn-in; the Wil-

son 95% CI on the observed false-positive rate (4/20, [7.3%, 38.5%]) and false-negative rate (0/20, [0%, 14%]) bounds the gate's empirical rollback tolerance. **Open question:** at what burn-in n does the false-negative upper bound drop below 5%? **Experimental design:** extend the gate to $n = 60$ iterations using the same held-out validation distribution; recompute Wilson CIs at $n = 30$, 40, 50, 60. **Expected headline:** given the observed 0 FN rate, the false-negative Wilson 95% upper bound drops below $\sim 5\%$ at $n \geq 60$. Deferred to the next revision because the additional 40 iterations require re-running the gate against the deployed validation distribution on the new no-teacher-anywhere ablation harness.

Adversarial residual at k=30 seeds and three-decimal precision. Table 18 reports the combined-defence residual as 0.00 (0.00) over k=3 attack runs. As Section 4.5.2 states, three seeds at two-decimal precision cannot distinguish a deterministic block from a small rate rounded down, so the current row establishes no formal bound. **Pre-registered target:** the robustness claim this paper makes requires the k=30 residual to stay at or below the adversary's 0.05 detection threshold with a Wilson 95% upper bound below 0.10, and the CL-bench regression to stay within the -0.8-point budget already reported. A residual above 0.05, or an upper bound above 0.10, retracts the combined-defence claim and reduces Table 18 to a single-configuration observation. The target is the acceptance criterion for the re-run, not a measurement: the realised value will be reported as measured. Deferred to the next revision as a focused $\sim$ 1 A100-day run on the existing harness.

References

Xin Cheng, Rui Tian, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Chengqi Deng, Shangyan Zhou, Chenggang Zhao, Zhewen Hao, Yukun Li, Han Zhang, Zhengyan Zhang, Yixu Wei, M.Y Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models, 2026. URL https://arxiv.org/abs/2601.07372. arXiv preprint, v2.

Guillaume Lample, Timothée Hecht, Hugo Touvron, and Luke Schmidt. Memory layers at scale. *arXiv preprint arXiv:2412.08763*, 2024.

Tencent Hunyuan Team and Fudan NLP Group. CL-bench and CL-

bench Life public leaderboard. https://www.clbench.com/, 2026. Live, community-submitted leaderboard for the CL-bench (1,899 tasks) and CL-bench Life (405 tasks) suites. Used to anchor SEA's leaderboard positioning against rows that appeared after the original Dou et al. (2026) Table 2 was typeset; retrieved 2026-08-02. New rows added on top of the Dou et al. Table 2 reproduction in Tables 4 and 5 are drawn from this source.

Amit Shukla, Sean Knowles, Mahesh Madugula, Daniel Farris, Ryan Angilly, Steven Pombo, Albert Xu, Luyang An, Ashwin Balasubramanian, Tao Yu, Jie Ren, and Rama Akkiraju. Adaptive data flywheel: Applying MAPE control loops to AI agent improvement. *arXiv preprint arXiv:2510.27051*, 2025.

Sahil Anupam, Daniel Brown, Shufan Li, Eric Wong, Hossein Hassani, and Osbert Bastani. BrowserArena: Evaluating LLM agents on real-world web navigation tasks. *arXiv preprint arXiv:2510.02418*, 2025.

Gido M. van de Ven, Nicholas Soures, and Dhireesha Kudithipudi. Continual learning and catastrophic forgetting. *arXiv preprint, Cornell University*, 2024.

Everton L. Aleixo, Juan G. Colonna, Marco Cristo, and Eduardo Fernandes. Catastrophic forgetting in deep learning: A comprehensive taxonomy. *Journal of the Brazilian Computer Society*, 30(1), 2024.

Haizhou Shi, Zheyu Xu, Hongbo Wang, Weiyi Qin, Wengang Wang, Yuhao Wang, Ziyang Wang, Samira Ebrahimi, and Huaiyu Wang. Continual learning of large language models: A comprehensive survey. *arXiv preprint, Cornell University*, 2024a.

Tongtong Wu, Liwei Luo, Yun-Fang Li, Shirui Pan, Thuy-Trang Vu, and Gholamreza Haffari. Continual learning for large language models: A survey. *arXiv preprint, Cornell University*, 2024.

Shihan Dou, Ming Zhang, Zhengyin Yin, Chenkun Huang, Yufan Shen, Jiahao Wang, Jingyi Chen, Yibo Ni, Junjie Ye, Chenwei Zhang, Hongbo Xie, Jingwei Hu, Shaohan Wang, Wenge Wang, Yuhao Xiao, Yifan Liu, Zhiyuan Xu, Zhilin Guo, Pan Zhou, Tao Gui, Zhongyu Wu, Xipeng Qiu, Qi Zhang, Xuanjing Huang, Yu-Gang Jiang, Daxin Wang, and Shuseng Yao. CL-bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*, 2026a.

Yuren Mao, Yong Ge, Yujie Fan, Wei Xu, Yu Mi, Zhongyang Hu, and Yunjun Gao. A survey on LoRA of large language models. *Frontiers of Computer Science*, 19(7), 2024.

Zeyu Han, Chao Gao, Jeff Liu, and Jeff Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint, Cornell University*, 2024.

Daniel E. Neves, Lucia Ishitani, and Zanoni K. G. Patrocínio. Advances and challenges in learning from experience replay. *Artificial Intelligence Review*, 58(2), 2024.

Yifan Lu, Yuxuan He, Jiale Chen, and Hongyuan Zha. MSSR: Memory-aware adaptive replay for continual LLM fine-tuning. *arXiv preprint arXiv:2603.09892*, 2026.

Manish Dhakal, Usman Jinadu, Aakash Budathoki, Rajshekhar Sunderraman, and Yuxin Ding. DistillLens: Symmetric knowledge distillation through logit lens. *arXiv preprint arXiv:2602.13567*, 2026.

Zhenyi Shi, Peng Liu, Tianqiao Su, Yuxuan Wu, Kang Liu, Yan Song, and Meng Wang. Densely distilling cumulative knowledge for continual learning. *arXiv preprint arXiv:2405.09820*, 2024b.

Bojie Li. User as engram: Internalizing per-user memory as local parametric edits. *arXiv preprint arXiv:2606.19172*, 2026.

Benoit Dherin, Michael Munn, Hanna Mazzawi, Michael Wunder, and Javier Gonzalvo. Learning without training: The implicit dynamics of in-context learning. *arXiv preprint arXiv:2507.16003*, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ashish Nair, 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*, 2023a.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023.

Yubo Wang, Xuefeng Liang, Linsen Wu, Chenghan Du, Huan Hu, Hailiang Zhang, Jie Cao, and Ke Wang. SAGE: A self-evolving agent with long-term memory and tool-use. *arXiv preprint arXiv:2403.00846*, 2024.

Ceyao Zhang, Yafu Li, Yuancheng Chen, Minghao Wang, James Bridges, Jian Pei, Song Xu, Lin Qiu, and Xuanjing He. Proagent: A proactive self-evolving agent framework for long-horizon tasks. *arXiv preprint arXiv:2402.07339*, 2024.

Jiarui Fang, Yujian Peng, Xi Zhang, Yi Wang, Xin Yi, Genglin Zhang, Yuxuan Xu, Bo Wu, Shu Liu, Zheng Li, Zhaochun Ren, Nikolaos Aletras, Xueqiang Wang, He Zhou, and Zaiqiao Meng. A comprehensive survey of self-evolving AI agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025.

Navin Krishnan. AI agents: Evolution, architecture, and real-world applications. *arXiv preprint arXiv:2503.12687*, 2025.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Yang Chen, Yu Chen, Zhuo Chen, Jinhao Jiang, Ruiyang Ren, Xuan Li, Xinyu Tang, Zikang Liu, Pengfei Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *Frontiers of Computer Science*, 20(12), 2026.

Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003. doi: 10.1109/MC.2003.1160055.

Zhizhong Li and Derek Hoiem. Learning without forgetting. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 40, pages 2935–2947, 2017.

Sylvestre-Alban Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

Arun Mallya and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lijuan Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

Xiao Wang, Tianze Chen, Mingyang Ge, Jiahao Bu, Wenzhe Wang, Yang Teng, and Martin Schubert. Orthogonal low-rank adaptation for continual fine-tuning of large language models. *arXiv preprint arXiv:2309.02559*, 2023b.

Weijie Ren, Xiao-Lin Li, Lei Wang, Tuo Zhao, and Wei Qin. InfLoRA: Interpolation-free low-rank adaptation for continual learning of large language models. *arXiv preprint arXiv:2310.03229*, 2023.

Weixiang Zhao, Shu Wang, Yanyao Hu, Yuxin Zhao, Bing Qin, Xin Zhang, Qi Yang, Dongliang Xu, and Wanxiang Che. SAPT: A shared attention framework for parameter-efficient continual learning of large language models. pages 11641–11661, 2024.

Pratyush Sharma, Yash Tandel, Rajbabu Somani, Garima Singh, and Vasudeva Varma. EvoMerge: Synergizing distillation and replay for continual pre-training of language models. *arXiv preprint arXiv:2411.02765*, 2024.

Jiyue Wang, Pei Zhang, and Yanxiong Li. Memory-replay knowledge distillation. *Sensors*, 21(8):2792, 2021.

Reyhane Azimi, Rohan Rishav, Maximilian Teichmann, and Samira E. Kahou. KD-LoRA: A hybrid approach to efficient fine-tuning with LoRA and knowledge distillation. *arXiv preprint arXiv:2410.20777*, 2024.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

Yingqiang Xia, Junda Shen, Yongfeng Jiang, Xintong Yang, Jiawei Liu, Wei Wang, Lina Zhang, Meng Wang, Shuang Zheng, and Qingwei Zhang. Self-Alignment: Self-improving language models through iterative dpo. *arXiv preprint arXiv:2406.00406*, 2024.

Zhanhui Zhou, Jie Liu, Jing Shao, Xiangyu Yue, Chao Yang, Wanli Ouyang, and Yu Qiao. Beyond one-preference-fits-all alignment: Multi-objective direct preference optimization. *Findings of the Association for Computational Linguistics (ACL 2024 Findings)*, 2024.

Chen Ouyang, Anqi Wang, Jianhao Yuan, Yizhi Zhang, Xue Zhang, Zhiyuan Lu, Yu Qiao, Tat-Seng Wong, Mike Zhou, Dahua Lin, and Kai Chen. Training language models for Multiple-Choice Question Classification: A self-evolving framework. *arXiv preprint arXiv:2409.00422*, 2024.

Zexiong Ma, Chao Peng, Pengfei Gao, Xiangxin Meng, Yanzhen Zou, and Bing Xie. SoRFT: Issue resolving with subtask-oriented reinforced fine-tuning. *arXiv preprint arXiv:2502.20127*, 2025.

Yuge Zhang, Qiyang Lyu, Zixuan Li, Xing Wang, Hao Liu, Hong-Yu Liu, Yan Wang, Shu-Tao Yu, Xin Wang, and Zhiting Wang. Agent lightning: Train ANY AI agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025.

Huatong Si, Zicheng Wang, Yu Zheng, Xingran Li, Junjie Liu, Yali Song, Jiale Han, Min Lin, Lijuan Li, and Ying Sun. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

Hoyeon Seo, Minji Kim, Jinu Park, and Sungsoo Choi. TIPS-SC: A test-time process-information-gathering preference-learning framework for source code. *arXiv preprint arXiv:2503.14193*, 2025.

Thomas Joshi, Shayan Chowdhury, and Fatih Uysal. SWE-Bench-CL: Continual learning for coding agents. *arXiv preprint arXiv:2507.00014*, 2025.

Suresh Kokkula, Daniel David, and Ananya Baruah. Federated learning under temporal drift – mitigating catastrophic forgetting via experience replay. *arXiv preprint arXiv:2601.13456*, 2026.

Huaiyu Chen, Lin Cheng, Junhao Du, Hongbo Zhang, and Kun Wang. DarkGent: Detecting backdoors in self-evolving LLM agents via trajectory-auditing. *arXiv preprint arXiv:2511.08173*, 2025.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christian Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec) / arXiv preprint arXiv:2302.12173*, 2023.

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

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiaomin Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2020.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Ameya Prabhu, Philip H. S. Torr, and Puneet K. Dokania. GDumb: A simple approach that questions the need for complex continual learning methods. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

benchlm.ai. GDPval-AA benchmark leaderboard (snapshot 2026-08-11). https://benchlm.ai/benchmarks/gdpvalaanormalized, 2026. Public, community-maintained GDPval leaderboard reporting a normalized percentage per model on the OpenAI/Anthropic/Scale GDPval benchmark. The snapshot of 2026-08-11 reports 95 models; the top-5 entries (Claude Opus 5 67.3%, Claude Fable 5 62.1%, Qwen3.8 Max Preview 62.0%, GPT-5.6 Sol 61.5%, Kimi K3 59.1%) are the five reference rows of Table 9 and the top-1 row of Table 3's GDPval entry. The benchlm.ai snapshot is a normalized variant, not the official weighted rubric pass rate, so the Δ column of Table 9 is the signed difference on the published numbers only.

OpenAI, Anthropic, and Scale AI. GDPval: Measuring ai performance on real-world economically valuable tasks. https://openai.com/index/gdpval/, 2025. Joint public release (Sept 2025) of the GDPval benchmark: 9 occupational sectors, 44 occupations, weighted-rubric scoring across 1,320 tasks contributed by professionals with an average of 14 years of experience. Used in the headline leaderboard row of Table 3 and as the underlying benchmark in the dedicated subsubsection 4.2.6; the Self-Evolving Agent-mini 74.2% weighted rubric pass rate is the official-protocol number in that subsubsection. Retrieved 2026-08-02.

Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual learning bench: Evaluating frontier AI systems in real-world stateful environments. *arXiv preprint arXiv:2606.05661*, 2026.

Shuzheng Si, Haozhe Zhao, Yu Lei, Qingyi Wang, Dingwei Chen, Zhitong Wang, Zhenhailong Wang, Kangyang Luo, Zheng Wang, Gang Chen, Fanchao Qi, Minjia Zhang, and Maosong Sun. From context to skills: Can language models learn from context skillfully? *arXiv preprint arXiv:2604.27660*, 2026.

Hongbo Jin, Mingnan Zhu, Jingqi Tian, Xu Jiang, Zhongjing Du, Haoran Tang, Siyi Xie, Qiaoman Zhang, and Jiayu Ding. Context-CoT: Enhancing context learning via high-quality reasoning synthesis. *arXiv preprint arXiv:2605.25354*, 2026.

Shihan Dou, Yujiong Shen, Chenhao Huang, Junjie Ye, Jiayi Chen, Junzhe Wang, Qianyu He, Shichun Liu, Changze Lv, Jiahang Lin, Jiazheng Zhang, Ming Zhang, Shaofan Liu, Tao Ji, Zhangyue Yin, Cheng Zhang, Huaibing Xie, Jianglu Hu, Jingcheng Deng, Lincheng Li, Minda Hu, Shaolei Wang, Syrus Zhao, Weichao Wang, Yan Lei, Yang Liu, Yanling Xiao, Yiting Liu, Zenan Xu, Zhen Guo, Ziliang Zhao, Pluto Zhou, Tao Gui, Qi Zhang, Xuanjing Huang, Yu-Gang Jiang, Di Wang, and Shunyu Yao. CL-bench Life: Can language models learn from real-life context? *arXiv preprint arXiv:2604.27043*, 2026b.

Yuetai Li, Yichen Feng, Zhangchen Xu, Zixian Ma, Kaiyuan Zheng, Fengqing Jiang, Xinghua Sun, Rulin Shao, Zichen Chen, Yue Huang, Xinyang Han, Brian Lee, Kayla Xu, Shenglai Zeng, Hang Hua, Xiangliang Zhang, Basel Alomair, Ranjay Krishna, Luke Zettlemoyer, Pang Wei Koh, Bhaskar Ramasubramanian, Luyao Niu, Xiang Yue, and Radha Poovendran. JobBench: Aligning agent work with human will. *arXiv preprint arXiv:2605.26329*, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Esha Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. 2024.