FROM PRACTICE TO MODEL WEIGHTS: CONSOLIDATING THE SKILL LIBRARY OF A SELF-IMPROVING LLM AGENT INTO LORA

ABSTRACT

LLM agents can improve from practice without weight updates. A harness extracts skills from failed attempts, tests them, and stores the useful ones in a skill library retrieved for later problems. Because the model never changes, every stored skill consumes context and must be read again on every problem. We study when such a RAG library should be consolidated into model weights. Our harness approves candidate skills through an acceptance gate and diversity-aware selection. A heuristic trigger, LNIS, built from retrieval and outcome statistics the harness already logs, decides when to consolidate, training a LoRA adapter on the retained skills and emptying the library. We compare it with two retrieval-only harnesses, with and without periodic skill management. On LiveCodeBench, consolidation cuts input context per held-out problem by a factor of 5.6–5.8 and practice tokens by 34–39%, with LoRA training already included, and 45–55% less practice wall-clock time with LoRA training included. Held-out accuracy is 19.8% for our approach against 17.0% for both baselines. On MATH-500, consolidation still uses the fewest practice tokens, though it takes 18–26% longer, since the library never grows large enough for resets to pay back their training time. Managing the library separately, without ever consolidating it, produces no held-out benefit, so the gain traces to consolidation itself.

1 INTRODUCTION

A language model agent can get better at a task simply by practicing it, without any change to its parameters. After a batch of attempts, a harness looks at what went wrong, extracts a reusable skill from the failure, tests whether the skill actually helps, revises it if it does not, and keeps the useful ones in a skill library that later problems can draw on (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024). The model itself never changes; everything the agent has learned lives in this external library instead.

That external library is convenient, but it is not free. Every skill the library keeps has to be re-injected on every future problem, so the library must be curated, not just accumulated, or it silently taxes every future prompt. And because the model remains frozen, that tax is paid again and again. The same library is re-read on every problem, and every new candidate skill is tested with the current library already sitting in its prompt, so both the curation problem and the reading cost grow together as the library grows. This raises two questions. First, of everything the agent tries during practice, what is actually worth keeping in the library, given that context is not free? Second, given that the library is not free either, is there a point at which what it holds should stop living in the prompt and start living in the weights? We call this second event "consolidation", and we ask when it should happen:

- **RQ1 What should the library retain?** Which of the candidate skills produced during practice are worth their context cost?
- **RQ2 When should the retained skills move into the weights?** Given recent retrieval statistics, when should the library be consolidated into a LoRA adapter?

Work that moves context into weights already exists, but it treats the context as a fixed document to be converted once. Doc-to-LoRA (Charakorn et al., 2026), Cartridges (Eyuboglu et al., 2025), and Skill-to-LoRA (Zhang & Qi, 2026) each take a context that is given in advance and convert it in a single, offline step. A practicing agent does not have that luxury. Its library is a moving target that keeps changing as practice continues, so the harness has to decide, over and over, what belongs in it and when enough has accumulated to justify writing it into the weights.

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

Our harness (Section 3, Figure 1) answers RQ1 at two points in a skill's life. When a skill is first proposed, an acceptance gate discards it outright if it repairs nothing, or if it causes more regressions than repairs, on the window of problems that produced it. Maximal Marginal Relevance (MMR) (Carbonell & Goldstein, 1998) then chooses among the survivors, scoring each by an adaptation of the Cost-Normalized Gain (CNG) of Zhang & Qi (2026), net improvement divided by added context, while penalizing similarity to skills already chosen. Later, once a skill is living in the library, a periodic management pass prunes it again, removing skills that have stopped helping and near-duplicates. RQ2 is answered by a new trigger we call LNIS (Local Net-Improvement Saturation). It watches four signals, computed from three retrieval statistics and the repair and regression counts, comparing the three most recent skill deployments against the three before them (Section 3.4), and when it decides the library has stopped paying for itself, the harness trains a LoRA adapter (Hu et al., 2022) on examples of the retained skills, written by the policy model itself, and empties the library so the cycle can begin again (Section 3.5).

To find out whether any of this is worth doing, we compare three variants. Baseline 1 simply accumulates skills and retrieves them, forever. Baseline 2 adds the periodic management pass. Our approach adds LNIS-triggered consolidation on top of that. Because each variant adds exactly one component to the last, Baseline 1 versus Baseline 2 isolates the effect of management on its own, and Baseline 2 versus the full variant isolates the effect of consolidation on its own.

What we find. Consolidation earns its keep once the practice pool is large enough to make the library expensive. On LiveCodeBench (Jain et al., 2025), our method needs 5.6–$5.8\times$ less input context per held-out problem and 34–39% fewer practice tokens than either baseline, with LoRA training already included, and 45–55% less practice wall-clock time with LoRA training included — this efficiency gain is our primary result (Section 5.1). Held-out accuracy is 19.8% for our method against 17.0% for both baselines, a small edge that Section 5.2 examines in detail. On MATH-500 (Lightman et al., 2024), our method is within one problem of the baselines on the 50 held-out problems and still uses the fewest practice tokens, with 3.1–$3.6\times$ less context, even though it takes 18–26% longer overall. Its 450-problem practice pool never lets the library grow expensive enough for consolidation to pay for itself. On RQ1, Baseline 2 matches Baseline 1 on both benchmarks, so pruning by itself does not move held-out accuracy. The efficiency gain appears only when consolidation is added to curation.

Our contributions are:

- A harness that learns through a skill library during practice and periodically consolidates the library into LoRA weights, resetting it after each consolidation.
- LNIS, a heuristic consolidation trigger computed from retrieval statistics the harness already collects.
- A three-variant comparison that separates the effect of skill management from the effect of consolidation.
- A stage-by-stage cost accounting showing where consolidation saves compute (candidate evaluation), where it adds compute (training and data construction), and when the trade-off favors it.

2 Related Work

These related studies bear on the two research questions from the introduction. An extended discussion of related work, including a comparison along the design choices of our harness, is in Appendix F.

Learning from experience in context. Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023) use feedback on an attempt to improve the next attempt at the same task. Other agents carry experience across tasks by storing skills, reflections, workflows, or playbooks and retrieving them later (Wang et al., 2024; Park et al., 2023; Zhao et al., 2024; Wang et al., 2025; Suzgun et al., 2025; Zhang et al., 2025). This is retrieval-augmented generation (Lewis et al., 2020) applied to the agent's own history. None of these systems changes the model, so the stored experience keeps growing and must be read again for every problem.

Learning from experience in weights. STaR (Zelikman et al., 2022) and SEAL (Zweiger et al., 2025) fine-tunes the model on its own successful reasoning traces, and rejection-sampling fine-tuning

(Yuan et al., 2023) follows the same pattern. These methods write experience into the weights directly, without an intermediate library, and are not a like-for-like baseline for our question. Our three variants hold the source of knowledge fixed, the same extractor-written skills, and vary only whether it is delivered in context or in weights. Trajectory fine-tuning changes the knowledge source (raw trajectories, not tested skills), the update frequency (STaR retrains from the base model in every outer iteration (Zelikman et al., 2022); LNIS fired twice per run), and the data volume, so a difference against it could not be attributed to consolidation. Frequent parameter updates are also costly and prone to interference (Li et al., 2026), and (Gou et al., 2026) show that direct fine-tuning on collected agent experience recovers only 3.8% of the gains of in-context learning, whereas distilling the in-context experience into weights retains at least 64.8%. Our harness shares this in-context-first design but differs in three ways: it distills a gated, curated skill library rather than raw interaction histories, it decides online when to distill, and it repeats the cycle with a library reset. Our harness instead tests skills in context first and trains only on those that survive curation.

Moving context into parameters. Doc-to-LoRA (Charakorn et al., 2026) maps a document to a LoRA adapter (Hu et al., 2022), and Cartridges (Eyuboglu et al., 2025) train a compact KV cache on a corpus, so that later queries need not reprocess the context; related methods distill prompts or skills into parameters (Askell et al., 2021; Snell et al., 2022; Zou et al., 2024; Charakorn et al., 2025; Lu et al., 2026; Yu et al., 2026). Skill-to-LoRA (Zhang & Qi, 2026) is the closest work to ours. It converts each skill in a fixed library of human-written `SKILL.md` documents into its own LoRA adapter in a single offline step. This same paper defines the Cost-Normalized Gain, which we adapt to score candidate skills (Section 3.2). In its Shared-LoRA control, a single adapter trained on pooled data from several skills performed worse than per-skill adapters. Our consolidation trains one adapter on all retained skills, which is closer to that control. All three methods take the context to be converted as given. An online library needs more than a one-shot converter. Ours adds a trigger that decides when to convert, a reset after each conversion so the cycle repeats, and a curation step that decides what enters the adapter.

When to consolidate. Continual learning studies how to acquire new information without losing old knowledge (Shi et al., 2025). Elastic Weight Consolidation (Kirkpatrick et al., 2017), for example, penalizes changes to parameters that were important for earlier tasks, but such methods assume known task boundaries. Closest to our RQ2, (Li et al., 2026) route incoming knowledge into an external memory and periodically consolidate high-value entries into the weights through supervised fine-tuning, and (Gou et al., 2026) distill an agent's in-context experience into its weights, retaining most of the in-context gains where direct fine-tuning on the same experience recovers little. Both consolidate factual memories or raw interaction histories. We consolidate procedural skills that have passed an acceptance gate, let LNIS decide online when to do so from retrieval statistics the harness already logs, and reset the library after each consolidation so the cycle repeats.

3 METHOD

3.1 OVERVIEW

The harness learns at two levels (Figure 1). At the harness level, it turns practice failures into skills, keeps the useful ones in an external skill library, and retrieves them for later problems, without changing the policy model. At the model level, it trains a LoRA adapter on the retained skills and empties the RAG library.

Practice proceeds one window of k problems at a time. The policy model first attempts the window with whatever RAG library it currently has. An extractor model then looks at what went wrong and proposes candidate skills from the failed trajectories; each candidate is tested on the same window, and a candidate that helps is kept while one that does not is sent back to the extractor with a critique. The skills that survive this filter are scored and selected, then added to the RAG library, where LNIS is watching a small set of retrieval statistics in the background. When those statistics tell LNIS the library has stopped paying for itself in context, the harness turns the surviving skills into training data, trains a LoRA adapter, and empties the library, so the cycle can start again with a clean slate. Figure 1 traces this loop end to end. Sections 3.2 and 3.3 describe what the library keeps (RQ1), and Sections 3.4 and 3.5 describe when and how it is consolidated (RQ2).

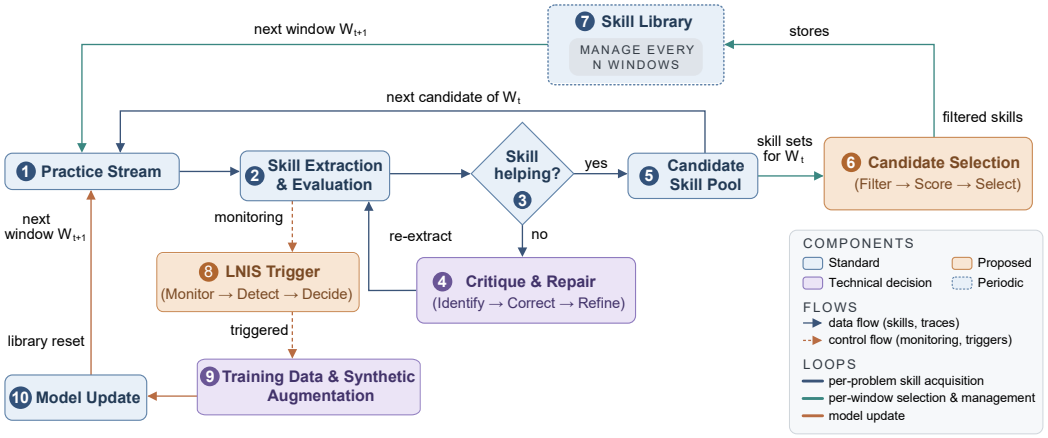


Figure 1: Overview of the harness. Practice failures are converted into skills at the harness level (steps 1–5). Useful skills are selected, stored, and retrieved for later problems (steps 6, 7). LNIS decides when the active skills are consolidated into a LoRA adapter at the model level (steps 8–10), after which the skill library is reset.

3.2 Skill Construction and Selection

Candidates. Let w_t be the t-th window and $\mathcal{L}_t$ the library when it starts. Each candidate skill has a trigger condition, a diagnosis, a numbered corrective procedure, a verification step, and a statement of when not to apply it. A structural filter rejects malformed candidates and candidates that contain problem-specific constants (Section B.2). A novelty check then compares each candidate with the library and with recently rejected candidates by embedding similarity. Candidates below 0.55 are novel, candidates above 0.85 are duplicates, and GPT-OSS-120B judges those in between (Section B.3).

Acceptance gate. Each novel candidate c is evaluated on the window that produced it. Let o_i^{base}, $o_i^c \in \{0,1\}$ be the outcomes on problem i with the current library alone and with c added. We count repairs (Rep) and regressions (Reg), and derive the net improvement NI and the activity A:

$$\begin{aligned} \mathrm{Rep}(c) &= \textstyle\sum_i \mathbb{1}\big[o_i^{\text{base}} = 0,\ o_i^c = 1\big], & \mathrm{NI}(c) &= \mathrm{Rep}(c) - \mathrm{Reg}(c), \\ \mathrm{Reg}(c) &= \textstyle\sum_i \mathbb{1}\big[o_i^{\text{base}} = 1,\ o_i^c = 0\big], & \mathrm{A}(c) &= \mathrm{Rep}(c) + \mathrm{Reg}(c). \end{aligned} \tag{1}$$

A candidate is accepted when it changes at least one outcome and does not do net harm:

$$\mathrm{NI}(c) \geq 0 \quad \text{and} \quad \mathrm{A}(c) > 0. \tag{2}$$

The gate is permissive. It accepts a skill that repairs one problem and breaks another ($\mathrm{NI} = 0$, $\mathrm{A} = 2$), and it measures both quantities on the same k problems that motivated the skill. It therefore removes skills with no effect on their source window; it does not certify that a skill generalizes. In our runs, top-ranked candidates often rested on a single repaired problem, and one targeted the answer format rather than the mathematics (Table 17). A rejected candidate is returned to the extractor with a critique. Each window keeps drawing candidates until 10 are accepted or 10 attempts are spent.

Selection. Among accepted candidates, the harness prefers skills that help at low context cost. Let T_{base} and T_c be the total input tokens of the window's problems with the current library and with c added. Zhang & Qi (2026) define the Cost-Normalized Gain (CNG) as a pass-rate change divided by the relative change in token cost. Their CNG divides the pass-rate difference against a no-skill baseline by $|\rho|/100$. We instead measure ρ against the current library, use net improvement as the gain, and floor the denominator at ϵ. We use the candidate's net improvement as the gain, bound the cost from below, and map the result to a quality score in $[0, 1)$:

$$\rho(c) = \frac{T_c - T_{\text{base}}}{T_{\text{base}}}, \qquad \mathrm{CNG}(c) = \frac{\mathrm{NI}(c)}{\max\big(\rho(c),\, \epsilon\big)^{\beta}}, \qquad Q(c) = \frac{\max\big(\mathrm{CNG}(c), 0\big)}{\max\big(\mathrm{CNG}(c), 0\big) + 1}, \tag{3}$$

with $\epsilon = 10^{-6}$ and $\beta = 1$. The floor ϵ gives the largest score to a candidate that adds no input tokens, so a candidate injected into few problems can score very high, since the score partly reflects how

rarely a skill is routed, not only how much it helps. The harness then selects skills greedily with MMR (Carbonell & Goldstein, 1998), trading quality against similarity to the set S already selected:

$$c^\star = \arg\max_c \Big[\, \lambda\, Q(c) \;-\; (1-\lambda)\, \max_{s\in S} \mathrm{sim}(c,s) \Big], \tag{4}$$

where $\mathrm{sim}(\cdot,\cdot)$ is the cosine similarity of sentence embeddings. We use $\lambda = 0.7$ and stop when the best remaining MMR score falls below 0.10. The selected set is evaluated jointly on the window and added to the library. We call a window in which at least one skill is added a *deployment window*.

3.3 Skill Retrieval and Management

For a problem p, the harness ranks skills by the similarity between each skill's trigger condition and the problem text. It injects, in rank order, the eligible skills

$$\mathcal{C}(p) = \big\{\, s \in \mathcal{L} \;:\; \mathrm{sim}\big(\mathrm{trigger}(s),\, p\big) \geq \tau_{\mathrm{route}} \big\}, \qquad \tau_{\mathrm{route}} = 0.10, \tag{5}$$

until they fill 80% of the context length W minus the problem's own length. A problem may therefore receive no skill, one, or several. Baseline 1 only adds skills. Baseline 2 and the full variant also run a management pass every 15 windows. The pass removes skills whose cumulative net improvement since acceptance is non-positive, and it removes the weaker skill of any pair whose embedding similarity is at least 0.85. A deployed set's repairs and regressions are split among its members in proportion to their injections, a heuristic, since no evaluation isolates individual members (Section B.6).

3.4 The LNIS Consolidation Trigger

LNIS (Local Net-Improvement Saturation) is a heuristic trigger evaluated at every deployment window. It keeps one record per deployment window since the last consolidation. Each record stores the newly deployed set's repairs, regressions, and net improvement, measured against the library before it, together with three retrieval statistics defined below. LNIS compares the most recent $H = 3$ records (the current region, subscript c) with the 3 before them (the previous region, subscript p). Sums over a region are written NI, Rep, and A; means over a region carry a bar. Three per-window statistics enter: $\overline{T}$, the mean input tokens per problem with the new set injected; $\overline{\kappa}$, the input tokens the new set adds divided by its number of injections (context cost per injection); and $\overline{f}$, the fraction of problems for which no skill clears τ_{route} (uncovered problems). The four signals are (more details are in section B.7)

$$\text{Saturation (Sat):} \quad \mathrm{NI}_c \leq 0,\;\; \mathrm{NI}_c < \mathrm{NI}_p,\;\; \mathrm{Rep}_c < \mathrm{Rep}_p,\;\; \mathrm{A}_c < \mathrm{A}_p, \tag{6a}$$

$$\text{Context pressure (CtxFull):} \quad \overline{T}_c \;\geq\; \tau_{\mathrm{ctx}}\, W, \tag{6b}$$

$$\text{Efficiency decline (EffDecl):} \quad \overline{\kappa}_c \;>\; \overline{\kappa}_p, \tag{6c}$$

$$\text{Ability decline (AbilDecl):} \quad \overline{f}_c \;>\; \overline{f}_p, \tag{6d}$$

with $W = 4096$ and $\tau_{\mathrm{ctx}} = 0.9$. Consolidation fires when any one of them holds. EffDecl and AbilDecl fire on any increase, with no minimum size of change.

Three rules limit how often LNIS can fire. First, the records are cleared at every consolidation, and every signal except CtxFull needs $2H = 6$ records, so at least six deployment windows must pass after a consolidation before Sat, EffDecl, or AbilDecl can be evaluated. Second, windows that deploy nothing add no record. Third, a trigger within three deployment windows of the previous one is deferred to the next deployment window. The baselines compute the same signals but never consolidate or clear their records. Because injection is capped at $0.8W$ minus the problem length, CtxFull can hold only when the problem text and prompt template push the total past $0.9W$. It acts as a backstop and never fired in our runs.

3.5 LoRA Consolidation

Training data. When LNIS fires, the harness builds a supervised dataset from the active skills. Each skill contributes up to 16 examples drawn from the window in which it was accepted. Successful trajectories come first. Failed trajectories come next, each paired with a hindsight correction that the

policy model itself writes when shown the problem, its failed attempt, the reference answer (empty for LiveCodeBench, where correctness is checked by running tests), and the skill. A correction is kept only if it passes the same grader used for practice. Any remaining slots are filled with synthetic examples, generated by having the policy model write a new problem that requires the skill and then solve it. Synthetic examples are not verified. Because a window has only k problems, at least $16 - k$ examples per skill are synthetic.

Training and reset. The harness trains a fresh rank-32 LoRA adapter (Hu et al., 2022) on top of the current model. Consolidations stack. The model with the first adapter merged in is the starting point for the second, and earlier training data are not replayed. The library is reset only if the new model without skills reaches at least 0.7 times the accuracy of the previous model with the library injected, measured on the 10 most recent practice problems. Otherwise the skills stay active and are consolidated again at the next trigger. The reset check compares against the previous model *with* the library, not against the previous model without skills, so it can pass an update that lowers no-skill accuracy. A separate before/after check of no-skill accuracy is logged but not enforced (table 18). At LiveCodeBench window 114, no-skill accuracy fell from 7/10 to 5/10 (Table 11), but the reset check passed ($5/10 \geq 0.7 \times 3/10$, table 10), so the update was kept.

4 Experimental Setup

We test the harness on two benchmarks chosen to contrast a large practice pool against a small one, then define the three variants from Section 1 precisely and describe how each is evaluated.

4.1 Benchmarks and Models

LiveCodeBench. We use all 1,055 code-generation problems of LiveCodeBench release_v6 (Jain et al., 2025), which covers problems released between May 2023 and April 2025. A random split (seed 0) holds out 106 problems (10%) and leaves 949 for practice. Qwen2.5-Coder was released in September 2024, and 713 of the 1,055 problems (about 68%) were released before then (release_v4 cutoff), so some may overlap its pretraining data.

MATH-500. A random split (seed 0) of the 500 problems of MATH-500, a subset of the MATH test set (Hendrycks et al., 2021) selected by Lightman et al. (2024), gives 450 practice and 50 held-out problems.

Models. The policy models are Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) for code and Qwen2.5-Math-7B-Instruct (Yang et al., 2024) for mathematics. GPT-OSS-120B (OpenAI, 2025) performs skill extraction, critique, and novelty judgment. The policy model writes all training data. Held-out problems are never used for practice, skill management, or consolidation. The one post hoc analysis that uses them is the sensitivity sweep in Section 5.3. All hyperparameters are listed in Table 18.

4.2 Variants

The three variants differ only in the components listed in Table 1.

Table 1: Components enabled in each variant.

Variant	Extraction and selection (§3.2)	Skill management (§3.3)	LoRA consolidation (§3.4–3.5)
Baseline 1	✓	×	×
Baseline 2	✓	✓	×
Full	✓	✓	✓

Each variant adds one component to the previous one. Baseline 1 versus Baseline 2 is therefore a clean test of skill management on its own, and Baseline 2 versus our proposed variant is a clean test of consolidation on its own. We use this pair, rather than a fixed consolidation schedule or a reset without training, as the comparison for consolidation. A fixed schedule needs two free choices, how many updates and when. Copying the LNIS windows would just borrow LNIS's own decisions, and any other choice would be arbitrary or need its own sweep. A reset without training discards the

skills and returns to the base model. We therefore do not add either as a fourth or fifth variant. For the same reason of comparability, we do not compare against fine-tuning directly on trajectories or against Skill-to-LoRA (Section 2).

4.3 Evaluation Protocol

Decoding and held-out evaluation. Practice trajectories use temperature 0.6 and nucleus sampling with top-p 0.95 (Holtzman et al., 2020). Candidate evaluation, set evaluation, and the consolidation checks use temperature 0. Each variant runs practice once. Each variant is then evaluated on the held-out set in its final state. The baselines use their final library. The full variant uses the model from its second consolidation together with the library accepted since then (6 skills on LiveCodeBench and 7 on MATH-500). Held-out generation uses temperature 0.6 and top-p 0.95, and we report the mean over three sampling seeds. Statistical tests are in Section A.2.

Timing and the merged adapter. In all runs of the full variant, generation ran with the newest LoRA adapter attached rather than merged. Keeping each adapter separate lets every consolidation be undone and inspected, and lets us measure the adapter's own cost directly. It slows decoding by 78–82% and adds 3.6 GB of GPU memory (Table 16). A deployed system would merge it, so numbers marked † are recalibrated for a merged adapter. Every generation that ran with the adapter is rescaled to the per-token speed measured before the first consolidation in the same run. The recalibration is insensitive to this choice. Rescaling with the directly measured overhead instead changes no practice total by more than 0.2 h (Section A.14). Token counts need no rescaling.

5 Results

Table 2 summarizes all variants. We report efficiency first, because it is where the variants differ most, and then held-out accuracy and practice-time behavior.

Table 2: Practice and held-out results. Practice accuracy is reported without and with skills; Gain is the difference in percentage points. Practice time and tokens cover all policy-model generation and extractor calls and, for the full variant, dataset construction and training for both consolidations. W/o skill is the base policy model's accuracy on the practice pool before any practice runs; held-out input tokens are the mean per problem. One practice run per variant. Best values in bold. †Estimated for a merged adapter (Section 4.3).

		Practice set					Held-out set		
Benchmark	Variant	W/o skill	With skill	Gain	Time (h)	Tokens	Accuracy	Latency (s)	Input tokens
LiveCodeBench	Baseline 1	16.82%	19.49%	+2.67	27.48	67.06M	16.98%	7.86	2990
	Baseline 2	16.82%	19.28%	+2.46	33.48	73.15M	16.98%	12.75	3057
	Full	16.82%	19.28%	+2.46	**15.09**†	**44.38M**	**19.81%**	**7.15**†	**530**
MATH-500	Baseline 1	68.59%	70.67%	+2.08	14.75	20.64M	62.00%	12.44	1944
	Baseline 2	68.59%	70.89%	+2.30	**13.80**	19.44M	62.00%	10.53	1694
	Full	68.59%	70.67%	+2.08	17.34†	**18.45M**	**64.00%**	**6.70**†	**546**

5.1 Efficiency

LiveCodeBench. The full variant completes practice in 15.1 h†, against 27.5 h for Baseline 1 and 33.5 h for Baseline 2, and uses 44.38M tokens against 67.06M and 73.15M. Both totals include both consolidations. Figure 2 splits the time by stage. Most of the saving is in candidate evaluation, the most expensive stage in every variant: 7.6 h† for the full variant against 15.8 h and 24.8 h. Each candidate is evaluated with the active library in its prompt, and consolidation keeps that library small (Figure 3, bottom). Training and dataset construction cost 1.9 h†, well below the evaluation time saved. For wall-clock time, Baseline 1 is the cleaner comparison. It and the full variant generated at the same per-token speed before the first consolidation (4.9 and 5.0 ms per output token), whereas the Baseline 2 run was about 35% slower per token (6.7 ms) for reasons unrelated to the method. Token counts do not depend on hardware and give the same ordering.

MATH-500. The full variant again uses the fewest tokens (18.45M against 20.64M and 19.44M), but it takes longer (17.3 h† against 14.8 h and 13.8 h). Its candidate evaluation takes 13.1 h† against

11.8 h and 10.8 h. It evaluated more candidates (516 against 485 and 482), and although its total candidate-evaluation tokens are lower (9.21M against 12.11M and 10.40M, table 5).

Training and dataset construction add 1.2 h†. On a 450-problem pool, the library never grows large enough for resets to pay for training.

Cost of consolidation. As a rough measure of amortization, Table 6 divides the size of the consolidation training data (characters divided by four, counted once) by the input tokens saved per problem (the full variant's mean input during practice set evaluation, with the library in context, minus its mean held-out input after consolidation). This *token-equivalent ratio* is not a compute break-even. It counts the training data once even though training runs three epochs, and it equates training tokens with inference input tokens. On LiveCodeBench, the 713,215 training-data tokens of the two consolidations equal the input saved on 412 problems (1731 tokens saved per problem); on MATH-500, the 307,917 tokens equal the input saved on 698 problems (441 per problem).

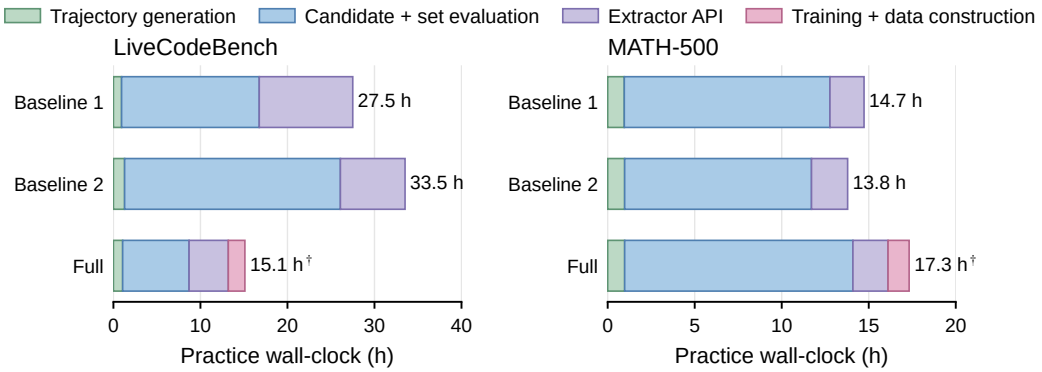


Figure 2: Practice wall-clock time by stage (one practice run per variant). Extractor API is GPT-OSS-120B extraction, critique, and novelty judgment. †Estimated for a merged adapter.

5.2 Held-Out Accuracy

LiveCodeBench. The full variant solves 21 of 106 held-out problems on average (19.81%, Wilson 95% interval over problems $[13.3, 28.4]$) and each baseline solves 18 (16.98%, $[11.0, 25.3]$). In the paired comparison with Baseline 1, 9 problems are solved only by the full variant and 6 only by Baseline 1 (Table 4). This gives an exact McNemar $p = 0.61$ and a 95% interval for the paired difference of $[-4.3, 10.0]$ percentage points. The full variant reaches this accuracy with 530 input tokens per problem instead of 2990 and 3057, and at the lowest latency of the three (7.15 s† against 7.86 s and 12.75 s).

MATH-500. The full variant solves 32 of 50 held-out problems (64.00%, $[50.1, 75.9]$) and each baseline solves 31 (62.00%, $[48.2, 74.1]$), a one-problem difference well inside both intervals. It uses 546 input tokens per problem instead of 1944 and 1694, a factor of 3.1–3.6 less. It again has the lowest latency of the three (6.70 s† against 12.44 s and 10.53 s).

Skill management alone (RQ1). Baseline 2 and Baseline 1 have the same held-out accuracy on both benchmarks. On LiveCodeBench they agree on all but 12 of the 106 problems, 6 in each direction (Table 4). In the full variant, management also determines which skills become training data, and our experiments do not separate that role from consolidation itself.

5.3 Practice-Time Behavior

All three variants reach similar practice accuracy (Table 2). Their net improvements over the practice pool are +24, +23, and +23 on LiveCodeBench and +9, +10, and +9 on MATH-500 (Table 12). Figure 3 shows how they get there. The baselines' libraries grow to 20–50 skills, while the full variant's library returns to zero at each consolidation. The full variant's cumulative net improvement keeps rising after both resets, so emptying the library does not stall practice.

Resets also change which candidates can pass the gate. On LiveCodeBench the full variant accepts 90 skills against 35 and 41 for the baselines (Table 13). It judges fewer candidates novel (322 against 483 and 452), but 28% of its novel candidates pass the gate, against 7% and 9%. At the first LiveCodeBench consolidation the mean prompt already filled 72.5% of W (Table 8). A possible explanation is that, in a large library, a new candidate is often crowded out or injected beside related skills, so its activity is zero and the gate rejects it. Each reset returns the library to a size at which new candidates are injected and can change outcomes.

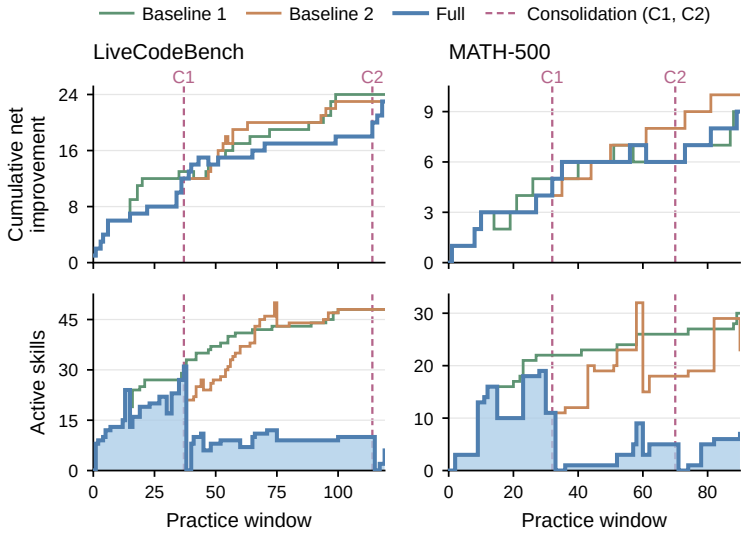


Figure 3: Cumulative net improvement (top) and active library size (bottom) over practice windows, one practice run per variant. Dashed lines C1 and C2 mark the full variant's LNIS-triggered consolidations; its library (shaded) is emptied at each one, while the baselines' libraries keep growing.

Which LNIS signals fired. EffDecl triggered three of the four consolidations and Sat the fourth (MATH-500, window 70); AbilDecl and CtxFull never fired.

Sensitivity. Varying β, λ, and τ_{route} one at a time on LiveCodeBench (Section A.3), the predefined default scored highest in each sweep; λ moves accuracy by about two points, while $\beta = 2$ and $\tau_{\text{route}} = 0.05$ lower it by more than five and more than six points, respectively. The sweep uses the held-out set, so it is a sensitivity analysis, not tuning.

6 Limitations and Conclusion

Limitations. Each variant runs practice once and the six runs in Table 2 already take over 120 GPU-hours, so three runs each would take more than two weeks on one GPU. Multi-seed sweeps of LLM training pipelines are widely regarded as infeasible at this cost (Vattikonda et al., 2025). We treat consistency across matched variants as partial evidence, not equivalent to seed replication.

Conclusion. This paper asked when a skill library, built and curated during an agent's own practice, should be written into the model's weights rather than carried in its prompt. Our answer is LNIS, a trigger built entirely from statistics the harness already collects, that decides for itself when the library has stopped paying for itself in context. On LiveCodeBench, where practice ran long enough for the library to grow expensive, consolidating on this trigger cut held-out context by 5.6–5.8× and practice tokens by 34–39%, an estimated 45–55% less practice time, without a measurable loss in held-out accuracy. On MATH-500, the same mechanism still found the cheapest path through the token budget, even though the library never grew large enough for training to pay for itself in wall-clock time. Pruning the library on its own, without ever consolidating it, changed neither result. The efficiency gain appears only when the library is consolidated, not when it is merely curated.

AI USE STATEMENT

Writing and presentation. Generative AI tools assisted with brainstorming, revising prose, checking terminology and cross-references, and preparing figures and tables. The authors reviewed the resulting text and visualizations, approved the final presentation, and take responsibility for every claim in the paper.

Literature discovery. Generative AI tools assisted with locating potentially relevant papers and organizing comparisons with related work. Citations and the claims attributed to them were checked against the cited sources rather than accepted from generated summaries.

Code and analysis support. Generative AI tools assisted with writing, debugging, and reviewing code used for data organization, analysis, and plotting. The authors specified the research questions, experimental variants, statistical quantities, and acceptance thresholds, and reviewed the reported outputs against the retained records. The authors determined the methodology, interpretation, and conclusions.

ETHICS STATEMENT

This work trains and evaluates language-model agents on public benchmarks (LiveCodeBench, MATH-500) and does not recruit or interact with human participants or analyze personal or sensitive data. Practice and consolidation require GPU compute; Sections A.4 and A.13 report the wall-clock and token cost of every run so it can be weighed against the reported savings. Roughly two-thirds of the LiveCodeBench problems predate the policy model's release, so some may overlap its pretraining data (Section 4.1); this affects all three variants equally but can inflate absolute accuracy, and should be accounted for before treating these numbers as an external benchmark score. The held-out accuracy comparison is not statistically significant at the reported sample size (Section 5.2); the principal practical risk we see is a reader treating the observed gain as an established improvement rather than as the open question Section 6 states it to be.

REPRODUCIBILITY STATEMENT

Section 3 gives the harness's method in full, including the acceptance gate, selection, routing, LNIS, and consolidation procedures, and Section B gives pseudocode and further implementation detail for each stage. Section C.2 lists every hyperparameter used in every run. The benchmarks, models, and evaluation protocol are in Section 4, and the statistical procedures behind the held-out comparisons, exact McNemar tests and Wilson intervals, are in Sections A.1 and A.2. Section D reports every ablation and sensitivity check we ran, including one still in progress, so that partial results are not omitted. Each variant in the main results is a single practice run with a fixed seed; Section 6 states this limitation and what repeating it would require.

REFERENCES

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

Jaime G. Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 335–336, New York, NY, USA, 1998. Association for Computing Machinery. doi: 10.1145/290941.291025.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-LoRA: Instant transformer adaption. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 7485–7514, 2025.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-lora: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*, 2026.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. doi: 10.48550/arXiv.1604.06174.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 10088–10115, 2023. doi: 10.48550/arXiv.2305.14314. arXiv:2305.14314.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study. *arXiv preprint arXiv:2506.06266*, 2025. doi: 10.48550/arXiv.2506.06266.

Chenhui Gou, Haoqin Tu, Yunhao Fang, Jianfei Cai, and Hamid Rezatofighi. Sample-efficient learning from agent experience. *arXiv preprint arXiv:2607.21051*, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. doi: 10.48550/arXiv.2103.03874. arXiv:2103.03874.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations (ICLR)*, 2020. doi: 10.48550/arXiv.1904.09751. arXiv:1904.09751.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. doi: 10.48550/arXiv.2106.09685. arXiv:2106.09685.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, Keming Lu, Fan Yang, Yichang Zhang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024. doi: 10.48550/arXiv.2409.12186.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations (ICLR)*, 2025. doi: 10.48550/arXiv.2403.07974. arXiv:2403.07974.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 9459–9474, 2020. doi: 10.48550/arXiv.2005.11401. arXiv:2005.11401.

Wenzhi Li, Dong Nie, Rui Lan, Tongtong Lyu, Peiyao Wang, Lingzi Hong, Weihang Pan, Boyuan Pan, and Yao Hu. Dual-layer agentic memory with fast write routing and slow consolidation. *arXiv preprint arXiv:2608.22215*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023. doi: 10.52202/075280-2019. arXiv:2303.17651.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Joon Sung Park, Joseph O'Brien, Carrie Jun 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)*, pp. 1–22, 2023. doi: 10.1145/3586183.3606763. arXiv:2304.03442.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410.

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. Continual learning of large language models: A comprehensive survey. *ACM Computing Surveys*, 58(5):1–42, 2025. doi: 10.1145/3735633.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 8634–8652, 2023. doi: 10.48550/arXiv.2303.11366. arXiv:2303.11366.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

Dheeraj Vattikonda, Santhoshi Ravichandran, Emiliano Penaloza, Hadi Nekoei, Megh Thakkar, Thibault Le Sellier de Chezelles, Nicolas Gontier, Miguel Muñoz-Mármol, Sahar Omidi Shayegan, Stefania Raimondo, Xue Liu, Alexandre Drouin, Laurent Charlin, Alexandre Piché, Alexandre Lacoste, and Massimo Caccia. How to train your LLM web agent: A statistical diagnosis. *arXiv preprint arXiv:2507.04103*, 2025. Accepted at NeurIPS 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning (ICML)*, 2025.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024. doi: 10.48550/arXiv.2409.12122.

Aofan Yu, Chenyu Zhou, Tianyi Xu, Zihan Guo, Rong Shan, Zhihui Fu, Jun Wang, Weiwen Liu, Yong Yu, Weinan Zhang, and Jianghao Lin. LatentSkill: From in-context textual skills to in-weight latent skills for LLM agents. *arXiv preprint arXiv:2606.06087*, 2026.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 15476–15488, 2022. doi: 10.52202/068431-1126. arXiv:2203.14465.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

Tianyi Zhang and Zhonghao Qi. Skill-to-lora: From using skills to learning behaviors for token-efficient llm agents. *arXiv preprint arXiv:2606.16769*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. arXiv:2308.10144.

Jiaru Zou, Mengyu Zhou, Tao Li, Shi Han, and Dongmei Zhang. PromptIntern: Saving inference costs by internalizing recurrent prompt during large language model fine-tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10288–10305, 2024. doi: 10.18653/v1/2024.findings-emnlp.602.

Adam Zweiger, Jyothish Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

APPENDIX

A Additional Results

Every table and figure behind the numbers claimed in Sections 3–5, ordered from the statistical validity of the headline result down to skill-lifecycle detail.

- A.1 Held-Out Results with Intervals — confidence intervals behind Table 2.
- A.2 Statistical Analysis — the McNemar test behind the held-out accuracy comparison.
- A.3 Sensitivity to Selection and Routing Parameters — the β, λ, τ_{route} sweep.
- A.4 Token Cost by Stage — the token totals behind Figure 2.
- A.5 Amortization — the token-equivalent ratio behind Section 5.1.
- A.6 LNIS Trigger Events — which signal fired at each consolidation.
- A.7 Consolidation Events — dataset size, steps, and loss for every consolidation.
- A.8 Adapter Versus Retrieval at Consolidation Checkpoints — the library-reset check.
- A.9 Accuracy Before and After Each Consolidation — where the before/after check was not enforced.
- A.10 Repairs and Regressions — practice-pool detail behind Table 2's Gain column.
- A.11 Candidate Funnel — extraction through acceptance, and novelty decisions.
- A.12 Skill Management Events — every pruning pass in Baseline 2.
- A.13 Unmerged Adapter Overhead — the 78–82% decoding cost of the unmerged adapter.
- A.14 Recalibration to a Merged Adapter — how every † number in the paper is estimated.

B Additional Method Details

Implementation detail for each stage of Section 3's pipeline, in the same order it appears there.

- B.1 Practice Loop — pseudocode for the full variant.
- B.2 Practice Windows and Skill Extraction
- B.3 Novelty Checking
- B.4 Cost-Normalized Gain and Selection
- B.5 Example Skills — two ranked candidates from each benchmark, with our critique.
- B.6 Skill Management
- B.7 LNIS Details

C Additional Experimental Details

How training examples are built, and the full configuration.

- C.1 Consolidation Data — how training examples are built.
- C.2 Hyperparameters — Table 18, the full configuration.

D Ablation Studies

Gathers every ablation and sensitivity check in one place: two full-scale component ablations already in the main text, a selection-mechanism pilot, and a hyperparameter sweep.

- D Overview — what each ablation isolates, and what is still in progress.
- D.1 Selection-Quality Pilot — randomly-selected skills are unanimously pruned; quality-selected skills are not.

E Held-Out Diagnostics

Four post-hoc checks on saved checkpoints; no new practice run.

- E.1 What the Consolidated Model Retains — remaining skills add nothing measurable.
- E.2 Forgetting Around Each Consolidation — zero problems lost, six events.
- E.3 Synthetic Data and Coverage — ties all-data at a fraction of the cost.
- E.4 A Paired, Three-Seed Comparison on MATH-500 — confirms the token reduction.

F Extended Related Work

- F Extended Comparisons — the related work of Section 2, discussed individually, plus Table 24.

A Additional Results

A.1 Held-Out Results with Intervals

Table 3 repeats the held-out columns of Table 2 with confidence intervals.

Table 3: Held-out results, averaged over three sampling seeds. Brackets give Wilson 95% intervals over problems; they ignore the pairing between variants and variation across practice runs. †Estimated for a merged adapter.

Benchmark	Variant	Accuracy	Input tokens	Latency (s)
LiveCodeBench	Baseline 1	16.98% (18/106) [11.0, 25.3]	2990	7.86
	Baseline 2	16.98% (18/106) [11.0, 25.3]	3057	12.75
	Full	**19.81% (21/106)** [13.3, 28.4]	**530**	**7.15**†
MATH-500	Baseline 1	62.00% (31/50) [48.2, 74.1]	1944	12.44
	Baseline 2	62.00% (31/50) [48.2, 74.1]	1694	10.53
	Full	**64.00% (32/50)** [50.1, 75.9]	**546**	**6.70**†

A.2 Statistical Analysis

Table 4 gives paired held-out outcomes on LiveCodeBench with the continuity-corrected McNemar statistic and the exact two-sided McNemar p-value. Each pair of variants agrees on at least 89 of the 106 problems, and no difference is significant at $\alpha = 0.05$. Per-problem held-out outcomes were not logged for MATH-500, so no paired test is possible there.

Table 4: Paired outcomes on the held-out LiveCodeBench set. "Only A" counts problems solved by the first variant and not the second.

Comparison	Both	Only A	Only B	Neither	χ^2	Exact p
Full vs. Baseline 1	12	9	6	79	0.267	0.61
Full vs. Baseline 2	11	10	7	78	0.235	0.63
Baseline 1 vs. Baseline 2	12	6	6	82	0.083	1.00

A.3 Sensitivity to Selection and Routing Parameters

After the main runs, we varied the CNG exponent β, the MMR weight λ, and the routing threshold τ_{route} one at a time, with all other settings, including the gate and LNIS, fixed at their defaults (Figure 4). This sweep was run on the held-out LiveCodeBench set, so it is a sensitivity analysis, not a tuning procedure. The defaults were fixed beforehand, but because the sweep shares its evaluation set with Table 3, it cannot rule out that the defaults happen to suit this particular set. In each sweep the default value scored highest. Accuracy varies by about two points across the tested values of λ (17.84% to 19.81%), whereas $\beta = 2$ lowers accuracy by more than five points and $\tau_{\text{route}} = 0.05$ by more than six. These drops exceed the gap between the full variant and the baselines, so the result depends more on β and τ_{route} than on λ. A sweep on a validation split carved from the practice pool, with several seeds, is needed to establish this.

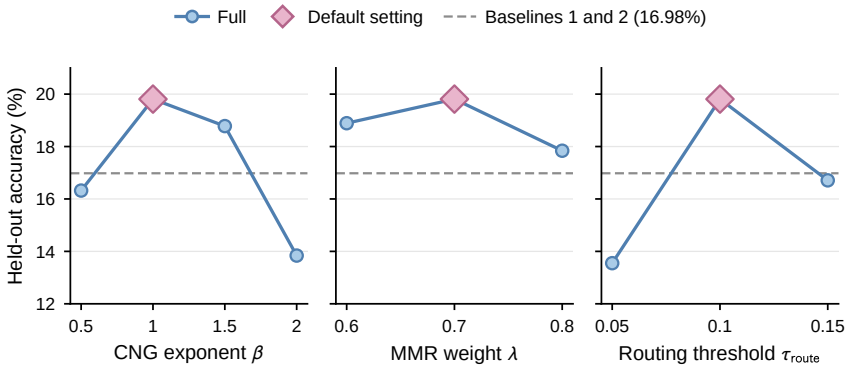


Figure 4: Held-out LiveCodeBench accuracy of the full variant as a function of the CNG exponent β, the MMR weight λ, and the routing threshold τ_{route}, with the other parameters and LNIS at their defaults. Diamonds mark the defaults used in the rest of the paper; the dashed line is the baselines' accuracy (16.98%).

A.4 Token Cost by Stage

Candidate evaluation is the largest stage in wall-clock time in every variant (Table 5), and the largest token cost in every variant except the full variant on LiveCodeBench, where extraction (23.69M) exceeds it (14.67M). On LiveCodeBench it accounts for 52.0% and 59.9% of the total for Baseline 1 and Baseline 2 and 33.1% for the full variant; on MATH-500, 58.7%, 53.5%, and 49.9%.

Table 5: Token cost by pipeline stage, in millions of tokens. Train build is the generation cost of constructing the datasets for both consolidations of the full variant; it is not the dataset size used in Table 6.

Benchmark	Variant	Traj. gen.	Cand. eval.	Skill eval.	Critique	Extraction	Novelty	Train build	Total
LiveCodeBench	Baseline 1	1.19	34.88	1.39	0.30	26.19	3.11	0	67.06
	Baseline 2	1.14	43.79	2.07	0.36	23.39	2.39	0	73.15
	Full	1.26	14.67	1.27	0.12	23.69	2.46	0.91	44.38
MATH-500	Baseline 1	0.51	12.11	0.31	0.23	6.95	0.53	0	20.64
	Baseline 2	0.52	10.40	0.30	0.22	7.50	0.51	0	19.44
	Full	0.50	9.21	0.38	0.23	7.21	0.38	0.54	18.45

A.5 Amortization

Table 6 gives the figures behind the token-equivalent ratio of Section 5.1.

Table 6: Context reduction and consolidation cost for the full variant, over both consolidations. Prompt sizes and savings are tokens per problem. Training-data tokens were logged per run, not per consolidation, and are apportioned to consolidations by their number of training examples. Token-equivalent problems is training-data tokens divided by tokens saved per problem; it is not a compute break-even. †Estimated for a merged adapter.

Benchmark	Retrieval prompt	After consolidation	Saved	Training + data (h)	Token-equivalent problems
LiveCodeBench	2261	530	1731	1.90†	412
MATH-500	987	546	441	1.21†	698

A.6 LNIS Trigger Events

Table 7 counts, for each run, the deployment windows at which the full trigger (Equation (6), any signal) held after the cooldown was applied. The baselines compute the same signals with the same cooldown but never act on them and never clear their records. Every baseline trigger therefore compares regions of a library that keeps growing, whereas each full-variant trigger clears the records and requires six new deployment windows before Sat, EffDecl, or AbilDecl can be evaluated again. The counts are therefore not comparable across variants. These rules explain the 77-window gap between the two LiveCodeBench consolidations (Figure 5). Only 27 of the 120 windows deployed a

skill set. After the reset at window 37, the six records needed to evaluate EffDecl accumulated only by window 64 (deployments at windows 39, 40, 43, 47, 51, and 64). The tokens per injection then fell from 61 to 20 over the next three deployments and rose again at window 114, where EffDecl fired. In the full variant, both LiveCodeBench triggers came from EffDecl. On MATH-500, the trigger at window 32 came from EffDecl and the trigger at window 70 from Sat, with all four sub-conditions of Equation (6a) holding. AbilDecl never fired on either benchmark, so these runs provide no evidence about it. CtxFull did not fire either, which is expected given the injection budget (Section 3.4).

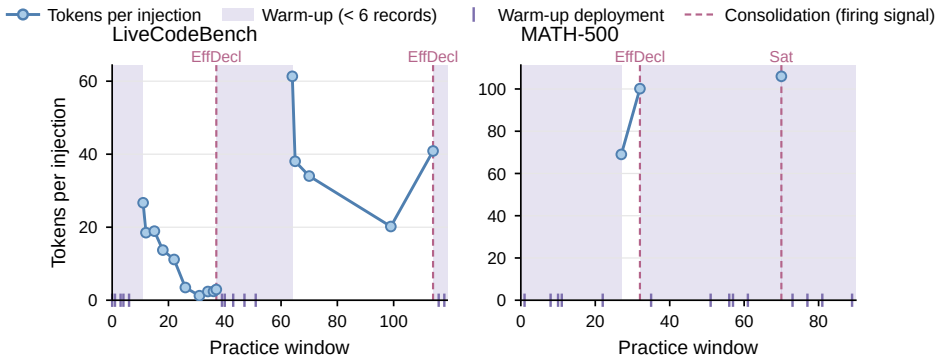


Figure 5: Tokens per injection of the current LNIS region at each deployment window of the full variant. LNIS needs six records to compare two regions, so after the start of practice and after each consolidation it is in warm-up (shaded) and cannot fire on EffDecl; the ticks on the axis are deployments made during warm-up, and a point appears once the sixth record exists. Dashed lines are consolidations, labelled with the signal that triggered them.

Table 7: LNIS trigger events over the 120-window LiveCodeBench run and the 91-window MATH-500 run. Mean gap is undefined for a single event.

Benchmark	Variant	Trigger events	First event	Mean gap (windows)	Consolidations
LiveCodeBench	Baseline 1	8	36	5.1	0
	Baseline 2	11	12	9.5	0
	Full	2	37	77.0	2
MATH-500	Baseline 1	0	n/a	n/a	0
	Baseline 2	1	73	n/a	0
	Full	2	32	38.0	2

Table 8: State at each consolidation in the full variant. Tok./inj. is the number of input tokens the deployed set added to the window divided by the number of skill injections in the window, averaged over the region. Context used is the mean input tokens per problem over the last three records as a fraction of $W = 4096$.

Benchmark	Window	Signal	Previous tok./inj.	Current tok./inj.	Context used	Skills consolidated
LiveCodeBench	37	EffDecl	1.22	2.92	72.5%	33
	114	EffDecl	38.03	40.85	50.4%	14
MATH-500	32	EffDecl	59.91	100.11	35.5%	19
	70	Sat	110.44	105.95	21.2%	7

A.7 Consolidation Events

Table 9 lists every consolidation. Each run has two consolidations, both triggered by LNIS. The RAG library remaining at the end of practice is not consolidated. On MATH-500 the final loss is higher than on LiveCodeBench at both events, and at window 70 it falls only from 1.491 to 1.358. The MATH-500 datasets are also smaller (304 and 112 examples against 528 and 224).

Table 9: LoRA consolidations in the full variant.

Benchmark	Window	Skills	Examples	Steps	Initial loss	Final loss
LiveCodeBench	37	33	528	99	0.798	0.430
	114	14	224	42	0.806	0.606
MATH-500	32	19	304	57	1.500	0.980
	70	7	112	21	1.491	1.358

A.8 Adapter Versus Retrieval at Consolidation Checkpoints

At each consolidation, the RAG library-reset check (Section 3.5) runs the updated model (newest adapter attached, unmerged) without skills and the pre-training model with the consumed skills injected on the 10 most recent practice problems. Table 10 reports these runs. With 10 problems, one problem is 10 points, so the accuracy columns cannot distinguish the two methods beyond passing the reset threshold (0.7), which every consolidation passed. Times are per problem with the adapter unmerged, and extra tokens are totals over the 10 problems. Retrieval adds 530 to 2,485 input tokens per problem, in line with the held-out difference between the baselines and the full variant on LiveCodeBench ($2{,}990 - 530 = 2{,}460$).

Table 10: Merged model without skills versus the pre-training model with the consumed skills injected, on the 10 most recent practice problems at each consolidation. Times are seconds per problem with the adapter unmerged; extra tokens are totals over the 10 problems.

Benchmark	Window	Active skills	Adapter acc.	Retrieval acc.	Adapter / retrieval time (s)	Extra retrieval tokens
LiveCodeBench	37	33	6/10	5/10	4.80 / 6.51	+24,853
	114	14	5/10	3/10	7.79 / 5.47	+21,212
MATH-500	32	19	9/10	9/10	6.18 / 6.19	+17,313
	70	7	7/10	7/10	5.42 / 3.82	+5,301

A.9 Accuracy Before and After Each Consolidation

Table 11 gives the accuracy without skills on the 10 most recent practice problems before and after each consolidation. Accuracy rose once, fell once (LiveCodeBench window 114), and was unchanged in the other two cases. This before/after check was logged but not used to reject updates. The enforced reset check (section A.8) passed at every consolidation, so the window-114 update was kept.

Table 11: Accuracy without skills on the 10 most recent practice problems, before and after each consolidation.

Benchmark	Window	Before	After
LiveCodeBench	37	5/10	6/10
	114	7/10	5/10
MATH-500	32	9/10	9/10
	70	7/10	7/10

A.10 Repairs and Regressions

Table 12 gives the practice-pool repairs and regressions behind the Gain column of Table 2.

Table 12: Repairs and regressions over the practice pool. Rates are relative to the pool size.

Benchmark	Variant	Pool	Repairs	Regressions	Net improvement	Repair rate	Regression rate
LiveCodeBench	Baseline 1	949	27	3	+24	2.85%	0.32%
	Baseline 2	949	26	3	+23	2.74%	0.32%
	Full	949	27	4	+23	2.85%	0.42%
MATH-500	Baseline 1	450	10	1	+9	2.22%	0.22%
	Baseline 2	450	10	0	+10	2.22%	0.00%
	Full	450	10	1	+9	2.22%	0.22%

A.11 Candidate Funnel

Table 13 follows candidates from extraction to acceptance, and Table 14 breaks down the novelty decisions.

Table 13: Candidate funnel over the LiveCodeBench and MATH-500 runs.

Benchmark	Variant	Extraction attempts	Safety passed	Judged novel	Accepted
LiveCodeBench	Baseline 1	3174	2534	483	48
	Baseline 2	2069	1652	452	76
	Full	4026	3080	322	90
MATH-500	Baseline 1	1251	920	485	29
	Baseline 2	1357	1053	482	54
	Full	1329	1012	516	67

Table 14: Novelty decisions over the LiveCodeBench and MATH-500 runs.

Benchmark	Variant	New	Duplicate	Too similar	Total judged
LiveCodeBench	Baseline 1	483	1764	287	2534
	Baseline 2	452	1041	159	1652
	Full	322	2225	533	3080
MATH-500	Baseline 1	485	143	292	920
	Baseline 2	482	361	210	1053
	Full	516	296	200	1012

A.12 Skill Management Events

Table 15 lists every management pass in Baseline 2, referenced in Section 3.3; Baseline 1 never prunes.

Table 15: Skill management passes for Baseline 2.

Benchmark	Window	Active before	Weak removed	Redundant removed	Active after
LiveCodeBench	14	24	11	0	13
	29	22	5	0	17
	44	27	3	0	24
	59	35	2	0	33
	74	50	6	1	43
	89	44	0	0	44
	104	48	0	0	48
	119	48	0	0	48
MATH-500	14	1	0	0	1
	29	9	0	1	8
	44	17	8	0	9
	59	29	12	5	12
	74	16	0	0	16
	89	26	1	5	20

A.13 Unmerged Adapter Overhead

Table 16 measures the decoding cost of an unmerged adapter after each consolidation, on 5 training prompts.

Table 16: Decoding overhead of the unmerged LoRA adapter at each consolidation checkpoint.

Benchmark	Window	Without adapter (s/token)	With adapter (s/token)	Overhead	Additional GPU (GB)
LiveCodeBench	37	0.00765	0.01390	+81.6%	3.60
	114	0.00770	0.01372	+78.0%	3.60
MATH-500	32	0.00747	0.01360	+82.0%	3.60
	70	0.00739	0.01344	+81.9%	3.60

A.14 Recalibration to a Merged Adapter

Keeping the adapter unmerged lets each consolidation be undone, as Section 4.3 explains. After the first consolidation, the per-token time of trajectory generation rises from 5.04 to 8.70 ms on LiveCodeBench and from 7.68 to 13.68 ms on MATH-500 (Figure 6), in line with the 78–82% overhead in Table 16. Baseline 1, which never trains, stays at 4.94 and 7.41 ms.

A merged adapter decodes at the base model's speed. We therefore multiply the time of every generation that ran with the adapter by

$$r = \frac{\text{s per output token before the first consolidation}}{\text{s per output token after it}},$$

measured on trajectory generation in the same run: $r = 0.579$ on LiveCodeBench and $r = 0.561$ on MATH-500. This applies to trajectory generation, candidate and set evaluation, and dataset construction after the first consolidation, and to held-out evaluation; LoRA training, extractor calls, and retrieval are not rescaled. The practice times in Table 2 sum all of these stages, and held-out latency falls from 12.34 to 7.15 s per problem on LiveCodeBench and from 11.94 to 6.70 s on MATH-500. Using the ratio from the direct overhead measurement instead (0.555 and 0.551) changes no practice total by more than 0.2 h and the held-out latency by no more than 0.3 s (to 6.85 s and 6.58 s); we report the more conservative values.

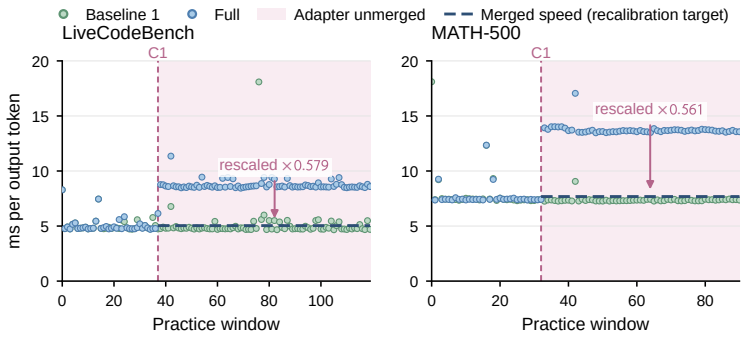


Figure 6: Per-token time of trajectory generation in each practice window. After the full variant's first consolidation (C1), generation runs with the unmerged adapter (shaded) and slows down, while Baseline 1, which never trains, keeps its speed. The dashed line is the full variant's pre-consolidation speed, the target of the recalibration; the arrow shows the factor r applied to generation time.

B Additional Method Details

B.1 Practice Loop

Algorithm 1 summarizes practice in the full variant. Baseline 2 runs the same loop without lines 19–24, and Baseline 1 also omits line 26.

Algorithm 1 Practice loop of the full variant.

Require: practice windows $w_1, \ldots, w_T$; policy model M; extractor E
$\mathcal{L} \leftarrow \emptyset$; $records \leftarrow [\,]$
for $t = 1$ **to** T **do**
  Attempt every problem in w_t with M, injecting skills from $\mathcal{L}$ (Equation (5))
  $\mathcal{A} \leftarrow \emptyset$
  for up to 10 extraction attempts, while $|\mathcal{A}| < 10$ **do**
    $c \leftarrow E$(failed trajectories of w_t and up to 64 recent trajectories) {≤ 3 tries}
    if c passes the structural filter and the novelty check **then**
      Evaluate c on w_t and compute $\mathrm{NI}(c)$, $\mathrm{A}(c)$ (Equation (1))
      if c passes the gate (Equation (2)) **then**
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{c\}$
      else
        Return c to E with a critique
      end if
    end if
  end for
  $S \leftarrow$ MMR selection from $\mathcal{A}$ (Equation (4))
  if $S \neq \emptyset$ **then**
    Evaluate S jointly on w_t; $\mathcal{L} \leftarrow \mathcal{L} \cup S$; append an LNIS record
    if LNIS fires (Equation (6)) and the cooldown has passed **then**
      Build the training set from the skills in $\mathcal{L}$; train a LoRA adapter on M; clear $records$
      if the reset check passes **then**
        $\mathcal{L} \leftarrow \emptyset$
      end if
    end if
  end if
  Every 15 windows: run the management pass on $\mathcal{L}$ (Section B.6)
end for

B.2 Practice Windows and Skill Extraction

For each window $w_t = \{p_1, \ldots, p_k\}$, the policy model attempts the problems with the current library $\mathcal{L}_t$, and the window's trajectories, together with up to 64 recent trajectories from earlier windows, are passed to the extractor. The extractor must propose a skill supported by at least two cited trajectories and return a trigger condition, a diagnosis, a numbered procedure of two to four steps, a statement of when not to apply the skill, a verification step, a statement of what the skill does not catch, and a one-line description. Each candidate then passes a structural filter, the "safety passed" stage in Table 13. The filter requires every field to be between 8 and 2,000 characters, at least two cited trajectory IDs that exist in the extraction pool, a numbered procedure, no problem-specific constants in the trigger or procedure (an assignment such as $x = 5$ or any number with three or more digits), and none of a list of prompt-injection phrases. Each extraction call gets up to three tries, shared between the structural filter and the novelty check, with the reason for failure passed back as feedback. A window makes up to 10 such attempts and stops early once 10 candidates are accepted. A candidate rejected by the gate is returned to the extractor with a critique of its repairs and regressions. Re-extractions after a gate critique are counted as additional extraction attempts in Table 13 and do not count toward the 10-attempt budget.

B.3 Novelty Checking

Each candidate c is compared with the current library $\mathcal{L}_t$ and the 10 most recently rejected candidates $\mathcal{R}_t$,

$$s(c) = \max_{u \in \mathcal{L}_t \cup \mathcal{R}_t} \mathrm{sim}\big(\mathbf{e}(c), \mathbf{e}(u)\big), \tag{7}$$

where $\mathbf{e}(\cdot)$ is the `all-MiniLM-L6-v2` sentence embedding (Reimers & Gurevych, 2019) of the skill's text fields and sim is cosine similarity. Candidates with $s(c) < \tau_{\mathrm{new}}$ are novel, candidates with $s(c) > \tau_{\mathrm{dup}}$ are duplicates, and candidates in between are judged by GPT-OSS-120B, which labels them new or too similar. Only novel candidates proceed to evaluation. Including recent rejections keeps the extractor from re-proposing a rejected idea when the library is empty, as it is at the start of practice and after each reset.

B.4 Cost-Normalized Gain and Selection

The relative token cost $\rho(c)$ (Equation (3)) is measured against T_{base}, the total input tokens of the window with the current library. T_{base} includes the problem text, so $\rho(c)$ is well defined when the

library is empty. In Equation (3), $\epsilon = 10^{-6}$ bounds the denominator when c adds almost no tokens, for example when it is injected into few problems. CNG is mapped to a bounded quality score,

$$Q(c) = \begin{cases} \mathrm{CNG}(c)/(\mathrm{CNG}(c)+1), & \mathrm{CNG}(c) > 0, \\ 0, & \text{otherwise}, \end{cases} \tag{8}$$

and MMR (Equation (4)) adds candidates to the selected set until the highest remaining MMR score is below 0.10. The first pick has no similarity penalty, so a single candidate is selected whenever $0.7\,Q(c) \geq 0.10$, i.e. $Q(c) \geq 0.143$.

B.5 Example Skills

Table 17 shows the two highest-ranked candidates from one window of each benchmark, by quality $Q(c)$ and by MMR score before the diversity penalty. In each window both candidates repaired the same single problem, so the gate and the quality score cannot tell them apart; the diversity term in Equation (4) is what separates them.

Table 17: The top two ranked candidates from one window on each benchmark, with their stored definitions and our assessment. Result gives repairs/regressions in that window, net improvement, the window's accuracy before and after, and how many problems in the window the skill fired on.

MATH-500

MATH-500 / Rank 1
Skill: *Canonical Form Verification*
Quality / MMR: 0.7114 / 0.4979

Trigger: When producing a final algebraic expression, check that it matches the exact expected canonical formatting. **Diagnosis:** Result is mathematically correct but fails because of extra spaces or non-canonical ordering. **Action:** Write the simplified expression in its most reduced symbolic form; remove unnecessary whitespace; order terms canonically; compare against the expected representation. **Verification:** String equality with the reference answer succeeds. **Description:** Focuses on matching exact textual representation, not merely mathematical value.

Result: 1 repair, 0 regressions, $+1$ net; $4/5 \rightarrow 5/5$; fired on $1/5$

Critique: Targets the exact-match grader rather than mathematical reasoning, so it likely encodes an evaluation artifact. Supported by a single problem.

MATH-500 / Rank 2
Skill: *Post-combination Verification*
Quality / MMR: 0.6992 / 0.4895

Trigger: After merging algebraic terms into a single expression, pause before finalizing. **Diagnosis:** The combined result may violate original constraints or contain sign or spacing errors. **Action:** Expand the derived expression back into its original components; simplify each; compare with the original terms; check signs and coefficients. **Verification:** Re-expanded form exactly reproduces the original separate terms. **Description:** A systematic back-check prevents silent algebraic mistakes after merging fractions or terms.

Result: 1 repair, 0 regressions, $+1$ net; $4/5 \rightarrow 5/5$; fired on $1/5$

Critique: Targets a real algebraic step, unlike #1, but is also supported by a single problem.

LiveCodeBench

LiveCodeBench / Rank 1
Skill: *Concrete Implementation Verification*
Quality / MMR: 0.9691 / 0.6784

Trigger: When the solution outline consists mainly of high-level description without actual code or detailed steps. **Diagnosis:** Missing concrete code implementation and edge-case handling. **Action:** Write the full function or program skeleton, including input parsing and output formatting; implement each step explicitly; add checks for empty, single-element, and extreme inputs; mentally test typical and edge cases. **Verification:** Answer includes runnable code that compiles or runs and passes an edge-case test. **Description:** Turns abstract plans into executable, edge-case-aware code.

Result: 1 repair, 0 regressions, $+1$ net; $0/8 \rightarrow 1/8$; fired on $1/8$

Critique: Specific and actionable, but fired on one problem and overlaps heavily with #2; the same repair is credited to several implementation skills.

LiveCodeBench / Rank 2
Skill: *Translate Reasoning into Concrete Code*
Quality / MMR: 0.9480 / 0.6636

Trigger: When the solution consists only of high-level description without actual implementation. **Diagnosis:** Missing concrete code leads to failure to compile or run. **Action:** Convert each algorithmic step into explicit statements or functions; write minimal I/O matching the problem format; add tests for empty, single-element, and typical cases; run and debug. **Verification:** Program runs without syntax errors and passes handcrafted edge cases. **Description:** Ensures the abstract plan becomes executable.

Result: 1 repair, 0 regressions, $+1$ net; $0/8 \rightarrow 1/8$; fired on $2/8$

Critique: Nearly a paraphrase of #1. Its pre-penalty MMR score is high, but the diversity term in Equation (4) lowers it once #1 is selected.

B.6 Skill Management

After each deployment window, the repairs and regressions of the deployed set are attributed to its members in proportion to how many problems each was injected into, and added to each member's

cumulative net improvement. In Baseline 2 and the full variant, a management pass every 15 windows removes every skill that has been in the library for at least one deployment and has cumulative net improvement at most 0. It then compares the surviving skills pairwise and, for each pair with embedding similarity at least 0.85 (the duplicate threshold of the novelty check), removes the one with lower cumulative net improvement. Table 15 lists every pass for Baseline 2. Baseline 1 never removes skills.

B.7 LNIS Details

LNIS keeps one record per deployment window since the last consolidation: the deployed set's repairs, regressions, and net improvement against the library before it; the mean input tokens per problem; the tokens per injection; and the fraction of problems with no injected skill. It is evaluated only when a record is added. CtxFull uses the mean input tokens of the last three records and can be evaluated from the first record onward. The other three signals need six records. Saturation (Equation (6a)) requires all four sub-variants at once, so a change in one statistic alone does not trigger consolidation. The fraction of problems with no injected skill counts only problems for which the library was non-empty.

Cooldown. The cooldown counts deployment windows. A trigger that holds within three deployment windows of the previous trigger is recorded as deferred and re-checked at the next deployment window. The counter starts at three, so the first trigger of a run is never deferred.

C Additional Experimental Details

C.1 Consolidation Data

Training examples are built per skill from the trajectories of the window in which the skill was accepted, up to 16 per skill, in this order. First come the successful trajectories the skill cites (all successful trajectories if it cites none), with the problem as prompt and the trajectory as completion. Next come the cited failed trajectories, each paired with a hindsight correction written by the policy model at up to 4,096 new tokens and kept only if it passes the grader. The prompt of a corrected example is the original problem, without the skill. Last come synthetic examples, generated one at a time by asking the policy model for a new problem that requires the skill, with a list of the problems it has already written so that each new one differs. The synthetic prompt includes the skill; synthetic examples are not verified. Since a window has k problems, at least $16 - k$ examples per skill are synthetic, at least 8 of 16 on LiveCodeBench and at least 11 of 16 on MATH-500. Every consolidation in Table 9 reached the cap of 16 per skill. On MATH-500 the logged composition was 52 successful, 21 corrected, and 231 synthetic examples at window 32; 13, 12, and 87 at window 70, so 76–78% of each dataset was synthetic. The composition was not logged for the LiveCodeBench runs.

Training uses QLoRA (Dettmers et al., 2023). The current model is loaded in 4-bit NF4 with double quantization and bfloat16 compute, a fresh LoRA adapter is attached to all linear layers, and the adapter is trained with the causal language modeling objective and a fresh optimizer for three epochs, capped at 200 steps, at an effective batch size of 16 (1×16 gradient accumulation) with gradient checkpointing (Chen et al., 2016). The adapter is then merged into the weights, and the merged model is saved as the starting point for the next consolidation. For generation during practice and held-out evaluation, we instead attached the newest adapter unmerged so that each consolidation could be undone (Section A.14).

C.2 Hyperparameters

Table 18 lists the configuration. All values were fixed before the main runs and are the same across benchmarks, except the window size k.

Table 18: Configuration used in all experiments.

Parameter	Value	Role
Practice and selection		
Window size k	8 (LCB), 5 (MATH-500)	Problems per practice window.
Candidate attempts / target	10 / 10 per window	Extraction budget and early stop.
Extraction tries per attempt	3	Shared by structural filter and novelty check.
Minimum cited trajectories	2	Structural filter.
Novelty thresholds $\tau_{\text{new}}/\tau_{\text{dup}}$	0.55 / 0.85	Novel / duplicate cut-offs.
Embedding model	`all-MiniLM-L6-v2`	Novelty, MMR, routing, and management.
CNG exponent β / floor ϵ	1.0 / 10^{-6}	Our adaptation of (Zhang & Qi, 2026).
MMR weight λ / stop threshold	0.7 / 0.10	Quality versus diversity; minimum MMR score.
Routing threshold τ_{route}	0.10	Minimum similarity for injection.
Injection budget	$0.8W$ minus problem length	Caps injected skill tokens.
Management period / duplicate threshold	15 windows / 0.85	Baseline 2 and Full only.
LNIS		
Region length H	3 deployment windows	Records per region.
Context length W / threshold τ_{ctx}	4096 / 0.90	CtxFull.
Cooldown	3 deployment windows	Minimum spacing of triggers.
Consolidation		
Examples per skill (cap)	16	Consolidation data.
Correction length	4096 new tokens	Hindsight corrections and synthetic examples.
Quantization	4-bit NF4, double quant., bf16 compute	QLoRA.
LoRA rank r / α / dropout	32 / 16 / 0.05	Adapter.
Target modules	all linear layers	Adapter.
Learning rate / schedule	10^{-4} / linear	Adapter training.
Epochs / step cap	3 / 200	Adapter training.
Effective batch size	16 (1×16 accumulation)	Adapter training.
Acceptance check	accuracy on 10 problems	Logged before and after; not enforced.
Reset check	ratio ≥ 0.7 on 10 problems	Merged model vs. retrieval before reset.
Decoding		
Practice / held-out	$T = 0.6$, top-p 0.95	Held-out: mean of 3 seeds.
Evaluation and checks	$T = 0$	Candidate, set, and consolidation checks.
Maximum new tokens	2048	Policy model generation.

D Ablation Studies

Three pieces of evidence bear on which components of the harness matter. Two are full-scale component ablations already reported in the main text: Baseline 1 versus Baseline 2 isolates skill management (Section 4.2), and Baseline 2 versus the full variant isolates consolidation (Section 5); both are evaluated on the complete practice pool and held-out set of each benchmark. Section D.1 adds a smaller, single-seed pilot that isolates the selection mechanism itself. Section A.3 is a sensitivity sweep over three selection and routing hyperparameters, not a component ablation, but included here for the same reason. A designed ablation of the library-reset gate (Section 3.5) is in progress; the runs completed so far have not yet included a training event for the gate to act on, so we do not report it here.

D.1 Selection-Quality Pilot

This subsection is a single-seed, 20-window pilot on MATH-500, run separately from the three variants in Section 4. It isolates one question: within the harness's RAG layer, does the quality-based scoring in Equations (3) and (4) pick better skills than an equally-sized random draw from the same accepted candidates would? It does not bear on RQ2, since LNIS needs six deployment windows since the last reset to evaluate three of its four signals (Section 3.4), and a 20-window pilot with one to two deployments per arm does not reach that count; no consolidation was triggered in any arm reported here.

Setup. Two arms share every practice window, extraction, and acceptance-gate decision up to the point where they select skills to deploy: *Quality*, which selects with Equation (4) as in the main runs, and Quality selected 4 skills and Random selected 6 from the identical accepted-candidate pool without MMR's stopping rule. The two arms are byte-identical through window 5; after that, sampling at temperature 0.6 lets their trajectories and candidate pools diverge, so sampling variance cannot be ruled out as a source of later differences.

Result. At the first deployment window (w12), Quality selected 4 skills and Random selected 6, each from its own pool of accepted candidates. Table 19 gives the joint evaluation on that window's 5 problems, at temperature 0, and the outcome of the next periodic management pass two windows later.

Table 19: Selection-quality pilot on MATH-500 (single seed, window 12 deployment, window 14 management pass). Joint evaluation accuracy is on the 5 problems of window 12, temperature 0.

Arm	Skills deployed	Joint eval. accuracy (no skill → with skill)	Net improvement	Survived w14 pruning
Quality (Equation (4))	4	0.20 → 0.40	+1	4 of 4
Random	6	0.20 → 0.20	0	0 of 6

Quality-selected skills repaired one problem and regressed none; randomly-selected skills changed no outcome on the same problems. Two windows later, the periodic management pass (Section 3.3) removed every one of Random's six skills as having non-positive cumulative net improvement, and kept all four of Quality's. The two checks are not independent. The pruning pass uses cumulative net improvement, which for Random's skills was 0 after the joint evaluation, so their removal follows directly from it.: the harness's own machinery treats the randomly-selected skills as not worth keeping, without being told which arm produced them.

Scope. This result is about what the RAG layer keeps, not about what reaches the adapter; Section 4.2 explains why we do not add further variants to test consolidation directly. We do not report raw window-by-window practice accuracy for this pilot: with only 4–6 skills ever active and 5 problems per window, it is dominated by which problems each window happens to contain rather than by which skills are active, and it is not a stable basis for comparison at this scale. A single seed and one benchmark are not enough to bound the effect size; the comparison is included as a directional check on the selection mechanism, not as a powered estimate of it.

E Held-Out Diagnostics

This appendix tests three concerns about the method on both benchmarks: what the consolidated model retains without its remaining skills (Section E.1), whether consolidation causes forgetting (Section E.2), and whether the synthetic examples are needed (Section E.3). Section E.4 adds a three-seed comparison on MATH-500. All checks reuse saved checkpoints; none required a new practice run.

E.1 What the Consolidated Model Retains

We compare the full variant's final checkpoint, with and without its remaining skills, against the base model, with and without those same skills. Sampling is at practice temperature (T=0.6).

Table 20: Held-out accuracy with and without the remaining skills (LiveCodeBench: 50 of 106 held-out problems, fixed random subset; MATH-500: all 50).

Benchmark	Model	No skills	With remaining skills	Remaining skills
LiveCodeBench	Base	16.0%	22.0%	6
	Consolidated	**24.0%**	22.0%	
MATH-500	Base	64.0%	54.0%	7
	Consolidated	**68.0%**	68.0%	

On MATH-500, the consolidated model scores exactly the same with and without its remaining 7 skills, 34 of 50 problems either way, the identical problems in both cases. On LiveCodeBench, the consolidated model without any skills reaches its highest accuracy in the table, 24.0%, above every cell that includes skills, base or consolidated. On both benchmarks, the remaining skills add nothing measurable to the consolidated model, and on LiveCodeBench they are mildly negative once the model has been consolidated. This is consistent with the held-out behavior living in the weights rather than being carried by what is left in the library. Neither difference reaches significance at this sample size (exact McNemar $p \geq 0.6$ throughout); the pattern is consistent across both benchmarks, which single-seed noise would not reliably produce in the same direction twice.

E.2 Forgetting Around Each Consolidation

For every consolidation, we evaluate the model immediately before and after, with no skills injected, greedy decoding, on up to 60 practice problems from windows already seen, split into those that fed the update (*since_prev*) and older ones that did not (*earlier*, the forgetting test).

Table 21: Problems lost (solved before, not after) on the *earlier* split at each consolidation. Zero problems were lost at any of the six consolidations checked.

Benchmark	Consolidation	Earlier problems	Solved before	Solved after	Lost
MATH-500	w70	28	20	20	0
	final	47	34	34	0
LiveCodeBench	w114	20	3	3	0
	final	57	16	18	0

Across four consolidation events on two benchmarks, not one previously-solved problem was lost. On LiveCodeBench, two of the three consolidations (w37 and final) show a net *gain* on problems the update was not trained on, with zero regressions; the third (w114) is exactly flat. MATH-500 is flat at every consolidation. We did not observe consolidation eroding earlier practice capability at any point we checked.

E.3 Synthetic Data Is Not Necessary Where Coverage Is Sufficient

For one consolidation on each benchmark, we rebuild the training set for a subset of its skills and train two adapters from the same starting checkpoint: *all-data*, the method as run, and *verified-only*, with every synthetic example removed.

Table 22: Verified-only vs. all-data. *Original* is the checkpoint produced by the full consolidation over all skills. Training examples and steps for the other rows are for the rebuilt subset.

Benchmark	Model	Solved	Train examples	Steps
MATH-500 (w32, 10 of 19 skills)	Before	43/60	—	—
	Original	42/60	—	—
	All-data	41/60	160	30
	Verified-only	41/60	35	9
LiveCodeBench (w37, 10 of 33 skills)	Before	18/60	—	—
	Original	17/60	—	—
	All-data	17/60	160	30
	Verified-only	18/60	24	6

On both benchmarks, removing every synthetic example costs nothing: verified-only matches all-data on MATH-500 ($41/60$ each) and is one problem higher on LiveCodeBench (18 vs. 17 of 60), while using 78–85% fewer training examples and 70–80% fewer steps. Neither adapter improves on the pre-update checkpoint for this subset ($43/60$ and $18/60$), so this ablation measures the cost of synthetic data rather than the benefit of consolidation, which Sections E.1 and E.2 address. The result is limited to skills with sufficient natural coverage: on MATH-500, the 9 skills excluded from this subset have fewer natural examples on average than the 10 included. For sparse skills, at least $16 - k$ examples per skill are synthetic by construction (Section 3.5), and we did not test whether removing them would hurt.

The rebuilt all-data adapter reproduces the original checkpoint within one problem on both benchmarks (41 vs. 42 of 60 on MATH-500, 17 vs. 17 on LiveCodeBench), which validates the rebuild procedure independently of the synthetic-data question.

E.4 A Paired, Three-Seed Comparison on MATH-500

Section 4.3 notes that only LiveCodeBench's held-out evaluation logged per-problem outcomes. We re-evaluate all three variants' final states on MATH-500 with three sampling seeds and per-problem logging, giving the paired comparison Table 2 could not.

Table 23: MATH-500 held-out accuracy, mean over three seeds, majority vote per problem.

Variant	Mean acc. (3 seeds)	Majority-vote correct	Input tok/problem
Baseline 1	63.3%	34/50	1944
Baseline 2	60.7%	31/50	1694
Full	63.3%	33/50	546

The single-run accuracy in Table 20 (68.0%) is higher than the three-run mean here (63.3%; individual runs: 68%, 68%, 54%) because a single run varies more; the three-run mean is the more reliable number.

The full variant matches Baseline 1's mean accuracy exactly and is within one problem of it on the majority vote (33 vs. 34 of 50, exact McNemar $p = 1.0$), and is numerically above Baseline 2 ($p = 0.625$). It reaches this with 3.1–3.6× less input context than either baseline, confirming the token reduction in Table 2 under an independent, multi-seed re-evaluation with per-problem outcomes.

F Extended Related Work

Section 2 discussed several lines of work in batches for space. This section describes them, and Table 24 compares the closest methods along the design choices of our harness.

Table 24: Comparison with the closest prior work. *Source*: what is converted or stored. *Evolving*: whether the stored knowledge changes during use. *Trigger*: what decides when knowledge moves into weights. *Reset*: whether the external store is emptied after conversion so the cycle repeats.

Method	Source	Evolving	Trigger	Reset
ExpeL, AWM, ACE	Insights, workflows, playbooks	Yes	Never converted	No
(Charakorn et al., 2026), Cartridges	Given document or corpus	No	One-time	n/a
(Zhang & Qi, 2026), LatentSkill	Given skill documents	No	One-time	n/a
(Lu et al., 2026)	Given skill documents	No	RL curriculum	n/a
(Gou et al., 2026)	Interaction histories	No	One-time	n/a
(Li et al., 2026)	Factual memories	Yes	Periodic	No
Ours	Gated, curated practice skills	Yes	LNIS (online)	Yes

Curated memory in context. Agent Workflow Memory (Wang et al., 2025) induces reusable workflows from past trajectories. Dynamic Cheatsheet (Suzgun et al., 2025) maintains an adaptive memory at test time, and ACE (Zhang et al., 2025) extends it into an evolving playbook maintained by separate generator, reflector, and curator roles, using incremental updates to prevent the context from collapsing under repeated rewriting. These systems curate what stays in context, which is the role skill management plays in our Baseline 2, but they never write the curated knowledge into the weights.

Self-generated weight updates. SEAL (Zweiger et al., 2025) lets a model generate its own fine-tuning data and update directives, trained with reinforcement learning on the performance of the updated model. Unlike our harness, it keeps no external library and has no decision about when to update.

Context distillation and internalization. Context distillation (Askell et al., 2021; Snell et al., 2022) fine-tunes a model to reproduce its own context-conditioned outputs without the context. Our consolidation data follows this principle: the policy model writes examples of each skill being applied, and the adapter learns to produce them without the skill in the prompt. PromptIntern (Zou et al., 2024) internalizes recurring prompts during fine-tuning, and Text-to-LoRA (Charakorn et al., 2025) trains a hypernetwork that maps a task description to a LoRA adapter.

Internalizing agent skills. Besides Skill-to-LoRA, two methods target agent skills. LatentSkill (Yu et al., 2026) trains a hypernetwork that compiles a textual skill into a LoRA adapter that can be loaded, scaled, and composed without retraining the backbone. SKILL0 (Lu et al., 2026) internalizes skills through reinforcement learning with a curriculum that starts with the full skill context and

progressively withdraws it. Both assume a skill library that exists before training begins, whereas ours is produced by the agent's own practice and keeps changing.

Further detail on the closest work. (Li et al., 2026) draw on Complementary Learning Systems theory. A cascade of small and large models labels each incoming knowledge item as not worth writing, new, or an update to stale knowledge, based on whether the model answers probes correctly with and without it. We instead admit a skill according to its measured repairs and regressions on the problems that produced it. (Gou et al., 2026) run their experiments on software-engineering tasks and text-adventure games and distill the interaction history once, whereas we consolidate repeatedly at points chosen by LNIS.