LEARNING IN LANGUAGE AND WEIGHTS: WHEN THE TWO REINFORCE EACH OTHER

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models are unique in that they can learn both from context and weight optimization, each with complementary strengths. Here, we study a method for combining them: the sample efficiency and interpretability of learning from natural language, and the permanence of weights learning. We show that CORE, a nonparametric learning algorithm, can produce natural language insights that can be directly distilled back into weights via *insight distillation*. Across six reasoning tasks and two models, we find that our method is more sample efficient and outperforms rejection-sampled fine-tuning on all 12 pairs, and tuned GRPO baselines on 10/12 pairs, and uses the distilled insights while solving problems. A further round of CORE learning improves on the previous round of insight distillation in 11/12 pairs, demonstrating a mutually reinforcing relationship between language and weights learning. Insight distillation is more interpretable than in-weights learning alone, and allows direct control on what a model learns: we are able to easily detect a reward hacking strategy occurring in the wild in a coding task, and remove the exploiting insights before distillation to prevent the behavior from forming. Finally, we show signs of life with an autonomous loop that can schedule repeated rounds of insight distillation for further learning efficiency. Overall, our results show that with powerful enough language models, learning in natural language and with weights can reinforce each other.

1 INTRODUCTION

A central challenge in machine learning is to design systems that learn from their own experiences rather than solely from curated demonstrations (Sutton & Barto, 2018). Humans naturally reflect on their experiences and consolidate them into insights on how to improve. With practice, such lessons become a durable skill, yet they can still often articulate what they learned. Reinforcement learning (RL) and similar methods allow language models to learn from their own attempts (Shao et al., 2024; Gulcehre et al., 2023) but are sample inefficient and reduce learning signal to scalar reward(s) (Agrawal et al., 2026; Melo et al., 2026), discouraging direct reflection on behavior changes. The lack of transparency in lessons learned during training raises concerns over interpretability and alignment (Hubinger et al., 2019; Baker et al., 2025). In context learning (ICL) (Brown et al., 2020) provides a way for models to learn directly in language via lessons or examples, but can not persist outside of context. A growing body of work uses language models to extract insights from their own experience (Shinn et al., 2023; Zhao et al., 2024; Ouyang et al., 2026; Nasvytis et al., 2026; Shalev et al., 2026), and some distill such experience or context into weights (Liu et al., 2026; Tiwari et al., 2026; Ye et al., 2026a; Snell et al., 2022; Lampinen et al., 2025). We present a method in which that consolidation transfers the in-context gain almost exactly, and that the distilled model retains the specific insights it was trained with: it recites them verbatim, and deleting insights before distillation removes the corresponding behavior from the weights.

We do this with an algorithm that alternates between the two systems. First, the model learns in language with CORE (Nasvytis et al., 2026). It contrasts failed and successful attempts on training problems, writes short insights, and draws on them in fresh attempts at the problems. Second, *insight distillation* consolidates the resulting insight bank into the weights using a form of context distillation in which the same model serves as both teacher and student. The two stages can then be repeated, with each round of learning in language starting from the weights produced by the previous round.

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

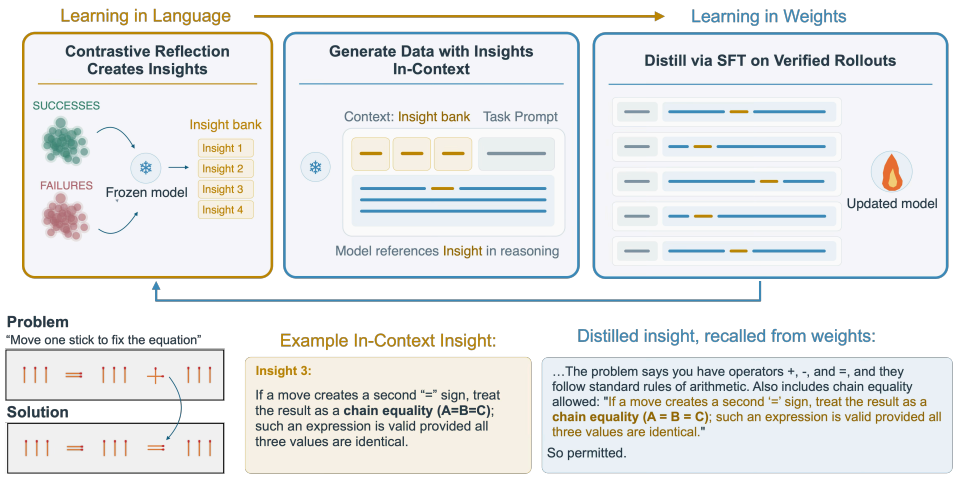

Figure 1: **Learning in language and in weights.** Learning in language (CORE) writes natural-language insights about the model's own attempts into a bank that is retrieved in context (weights frozen). Insight distillation samples solutions with the insights in the prompt and fine-tunes on them with the insights removed. The next round learns in language on the new weights.

Across six reasoning tasks and two models, gpt-oss-120b (OpenAI, 2025) and Qwen3.8-27B (Qwen Team, 2026), we show that insight distillation successfully retains model-written insights: the distilled model's gain over the base model is 1.02 times the gain of its insights in context, and on three of the six tasks it recites at least one insight verbatim in 2–39% of held-out rollouts. We also find it is far more sample efficient than tuned GRPO: at matched rollout budgets, it outperforms GRPO on 10 of 12 model–task pairs, and on 7 of them GRPO does not catch up even within five times the rollouts. It also outperforms rejection-sampled SFT on all 12, trading time spent reflecting for greater performance from few training examples. Continued rounds of CORE on the same data produce further gains, demonstrating how the distillation reinforces the initial insight production step.

Because the improvements come from short natural-language insights, what the model learns can be inspected and edited before it reaches the weights. We discover a reward hack in the wild in CrossPL (Xiong et al., 2026), a coding benchmark whose grader passes any program that exits with code 0. Models trained on this task learned this flaw, but removing offending insights before distillation prevented the model from consolidating and using this strategy at evaluation time.

Finally, we develop an autonomous loop that decides from training signals alone when to distill and when to stop. It matches the fixed two-round schedule on 11 of 12 pairs with 12% fewer rollouts, and its stopping rule fires after two to four rounds of insight distillation, once most of the improvement has ended. Our results demonstrate how learning from language and learning in weights can reinforce each other, and present an algorithm for retaining the strengths of both.

Our contributions are as follows:

- We show that in-context gains from self-written insights can be distilled and retained in the weights and are much more sample efficient than conventional learning systems. We find virtually zero negative off target effects on other tasks.
- We find that insight distillation can match or exceed tuned GRPO baselines, but can be complementary. At a matched number of rollouts, insight distillation is dramatically more efficient, but also provides a strong warm start for further RL training.
- Because insights are in natural language, we can audit insights and edit what the model learns. We find a grader-exploiting reward hacking behavior in the wild due to the exploit being written as an insight. We remove the exploiting insights prior to distillation and find the final model no longer exploits the hack, while also improving on the base model.
- We introduce an autonomous loop that decides from training signals alone when to distill and when to stop, matching a human-tailored training schedule with fewer rollouts.

2 Related Work

Learning in natural language. A growing line of research has explored how to improve the performance of large language models (LLMs) without changing their weights: From failed attempts (Shinn et al., 2023), via GEPA conditioned on rollouts (Agrawal et al., 2026), through a reasoning bank (Ouyang et al., 2026), and CORE extracts insights from contrasts between successful and failed attempts, admits only those that help on new attempts, and updates their utility by testing them on different problems (Nasvytis et al., 2026). We use CORE as the method for learning in language.

Distilling context into weights. Context distillation approaches are commonly used to distill context information into model weights (Snell et al., 2022). For example, this extra context can be externally generated or extracted from the model's past solutions (Ye et al., 2026b; Liu et al., 2026; Lampinen et al., 2025; Bai et al., 2022; Guan et al., 2025) or from self-training, where the model samples its own attempts and fine-tunes on the correct ones (Zelikman et al., 2022; Yuan et al., 2023; Singh et al., 2023). Our method, insight distillation, combines the two: it fine-tunes the model on its verified solutions, which were generated with self-written, individually tested insights in each prompt.

Combining learning in language and in weights. In-context and in-weights learning generalize differently (Lampinen et al., 2025; Russin et al., 2025; Anand et al., 2025), and pairing a fast and a slow learner has been studied from fast and slow weights in neural networks (Hinton & Plaut, 1987) to complementary learning systems in the brain, in which prior knowledge in turn speeds new learning (McClelland et al., 1995; Kumaran et al., 2016). Recent methods combine the two by alternating prompt or context learning with weight updates (Soylu et al., 2024; Tiwari et al., 2026; Ye et al., 2026a; Behrouz et al., 2026). We show that the two kinds of learning reinforce each other in both directions: insight distillation beats GRPO at equal rollouts, and learning in language improves again on the distilled weights.

3 Methods

We study two ways that a model can learn from its own attempts on a task (Figure 1): *learning in language*, where the weights stay frozen and the model writes natural-language *insights* into a bank, and later retrieves them in context (Section 3.1), and *learning in weights*, which we use to distill an insight bank into the model (Section 3.2). In terms of complementary learning systems (McClelland et al., 1995; Kumaran et al., 2016), the first is the fast, episodic system, the second the slow one, and insight distillation plays the role of systems consolidation. A *round* is a language phase followed by insight distillation. Throughout, *bare* means read without insights in the prompt.

3.1 Learning in Natural Language (CORE)

For learning in natural language, we use the CORE algorithm (Nasvytis et al., 2026). A CORE run starts with ten bare passes over the train set to estimate each problem's solve rate, and then repeats one training step until 1k attempts and admission trials have been made. In each step, the model attempts one training problem with retrieved insights in the prompt; problems are sampled with probability that increases in how often the model fails them, with a uniform share of $\epsilon_{\text{mix}} = 0.5$. Only failed attempts trigger reflection: the model then contrasts the failed attempt with the most recent correct attempt on the same problem, or on the most similar training problem if it has none, and writes up to three insights of at most 50 words. Each insight is a strategy, check or constraint that distinguishes the failed and the successful attempt. A candidate is admitted to the insight bank only if the model correctly solves the failed problem with that insight alone in the prompt. After each training attempt, every retrieved insight is credited $r - \hat{p}$, the attempt's reward minus the problem's solve rate. An insight's utility is its mean credit. Retrieval is k-nearest-neighbour over problem embeddings; with the neighbourhood set to the whole training split (as in every run here) it places the same 25 highest-utility insights in the prompt of every problem, in training and on held-out problems. The run's output is its final insight bank.

3.2 Learning in Weights (Insight Distillation)

Insight distillation moves what a bank teaches into the weights, in two steps. *Harvest:* the model attempts each training problem 16 times with the bank's top-25 insights in the prompt. *Fine-tune:* correct attempts become supervised targets, paired with the bare prompt. The model learns to produce, without the insights, the solutions it wrote while reading them. This is a form of self-distillation, close to context distillation (Snell et al., 2022): the same model is the teacher, with the insights in its prompt, and the student, without them. Unlike logit distillation, the student learns from the teacher's sampled and verified solutions, as in rejection-sampling fine-tuning (Yuan et al., 2023), STaR and ReST^{EM} (Zelikman et al., 2022; Singh et al., 2023). The whole accepted pool is used, without per-problem balancing. We fine-tune a rank-32 LoRA adapter (Hu et al., 2022) for 200 updates at a constant learning rate (batches of 16 rows for gpt-oss-120b and of 16 packed 34,816-token sequences for Qwen3.8-27B), with the loss on the response only; there is one learning rate per model, chosen on MathGAP and Matchstick and frozen before the other tasks were run (Appendix B). We call the insight-distilled model *CORE+SFT* in figures and tables.

3.3 Combining Learning in Language and Weights

We write A→B for method B started from the weights that A produced, and combine the two kinds of learning in two ways.

Alternating rounds of learning in language and in weights. A second round runs CORE on the round-1 CORE+SFT weights from an empty bank (*CORE+SFT→CORE*) and distills the new bank into round-2 CORE+SFT weights. The round-2 fine-tune continues the round-1 adapter with a fresh optimizer and the same learning rate.

Insight distillation, then reinforcement learning. *CORE+SFT→GRPO* runs GRPO (Shao et al., 2024) from the round-1 CORE+SFT weights for 40 updates with the recipe of plain GRPO (Section 3.4), on bare prompts.

3.4 Baselines

R-SFT denotes rejection-sampled fine-tuning on 16 bare attempts per training problem, sampled from the same starting weights, with the same acceptance rule and the same 200 updates. It isolates the benefit of generating training solutions with insights. **CORE extended** continues the round-1 CORE run on the base weights for as many rollouts as the insight distillation method spends on the harvest and the second CORE run (matched within 5%); this enables us to test whether learning for longer in language can improve as much as insight distillation at the same cost. **Plain GRPO** uses the base model on the same training problems, with 16 prompts $\times$ 16 samples per update, group-mean-centered advantages without standard-deviation normalization, no KL or entropy terms, rank-32 LoRA adapter, and 80 updates. On Qwen3.8-27B, standard GRPO collapsed or did not learn on Matchstick and Countdown at every learning rate we tried. Of the solutions we tested, only clamping negative advantages to zero led to stable training on both tasks, so Qwen3.8-27B uses this positive-only version on every task; Appendix C documents the 187 GRPO runs in this search.

3.5 Cost, Tasks and Evaluation

Cost. Costs are counted in rollouts, which are defined as any sampled attempt that produces a training signal. These include baseline passes, CORE attempts, admission trials, reflections, harvest attempts (whether or not accepted), and every GRPO sample (256 per update). Fine-tuning and evaluation generate no additional training rollouts; Appendix G reports the same comparisons in total generated tokens. A single insight distillation round costs 2.8–4.0k rollouts per task. A start from round-1 weights means being charged the round-1 cost.

Tasks. We use seven tasks with frozen training and held-out splits of 100 problems (except 60 for MathConstruct; the CrossPL training split has 99): MathGAP (arithmetic word problems), Matchstick (Roman equation puzzles solved by moving matchsticks), Countdown (reach a target number from given numbers), Chess960 (best move in a Fischer-random position), Enigmata BIG-Bench Symbolic (inferring a list-to-list function from input–output examples), CrossPL (Python programs

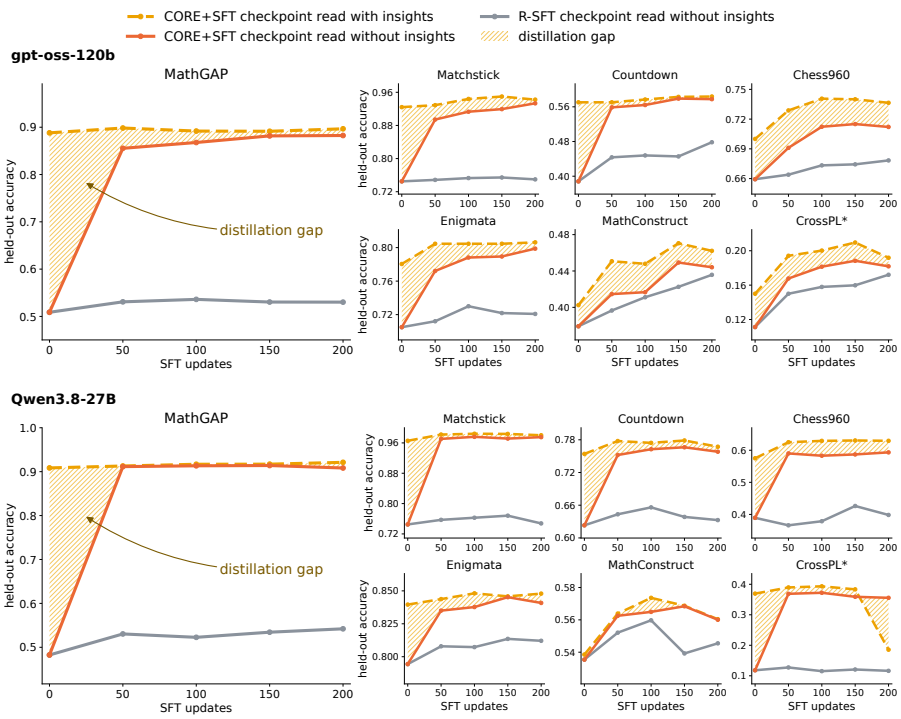

Figure 2: **Insights in context become weights.** Held-out accuracy of the fine-tuning checkpoints, read without insights (CORE+SFT, solid orange) and with the round-1 bank (dashed amber); gray: R-SFT, read without insights. Hatching: the gap between the two reads. Mean of three seeds.

that call C libraries) and MathConstruct (mathematical constructions checked by a verifier). CORE, harvests, GRPO all have the same grader in each task.

Models. gpt-oss-120b at high reasoning effort, temperature 1 and top-p 1, and Qwen3.8-27B at low reasoning effort, temperature 1, top-p 0.95 and top-k 20, following its model card; higher efforts truncated too many Qwen generations. Held-out reads cap generations at 29k tokens for gpt-oss-120b and 32,768 for Qwen3.8-27B (Appendix B lists every phase).

Evaluation. Held-out accuracy is measured as avg@16 (share of correct answers over 16 samples per problem), averaged over problems and read bare unless stated. Each result, unless otherwise stated, is the mean over three training seeds with ± 1 standard error. CrossPL*, whose original grader we found to be exploitable during our experiments (Section 4.5), is shown as recorded in the figures but excluded from every count, so counts are over the other 12 model–task pairs (Section 4.5).

4 Results

4.1 Language Insights Become Weights Without Losing Their In-Context Gain

The first question we ask is: can natural-language insights be distilled into the weights without losing their in-context gain?

CORE improves both models in-context. One CORE run (with the weights frozen) raises held-out accuracy by an average of 15.8 pp over the 12 model–task pairs, and by up to 42.6 pp. Appendix A shows CORE gains on each task per model.

Insight distillation keeps the in-context gain. If the insights are distilled successfully, the distilled model should reach the same performance gains without insights as it would with them in context. We find that the distilled model indeed reaches its in-context accuracy (Figure 2): across the 12 pairs, its gain over the base model is 1.02 ± 0.02 times the in-context gain, with a mean absolute difference of 1.2 pp. No pair ends more than two standard errors below its in-context accuracy, three

	CORE (no weights updated)	CORE extended	CORE+SFT→CORE	Error reduction
Qwen3.8-27B				
MathGAP	90.9% ±1.0	90.4% ±0.7	**93.5% ±0.8**	32%
Matchstick	96.6% ±0.7	96.1% ±0.9	**98.8% ±0.3**	69%
Chess960	57.5% ±1.6	60.7% ±2.2	**61.8% ±1.6**	3%
Countdown	75.4% ±0.5	73.9% ±1.1	**78.9% ±1.4**	19%
Enigmata	84.0% ±0.4	83.2% ±0.6	**83.4% ±0.5**	1%
MathConstruct	53.9% ±1.1	53.3% ±0.9	**56.7% ±0.4**	7%
gpt-oss-120b				
MathGAP	88.8% ±0.6	88.4% ±0.3	**89.2% ±0.7**	7%
Matchstick	92.4% ±0.6	93.3% ±0.8	**94.9% ±0.7**	24%
Chess960	70.0% ±0.7	70.5% ±1.0	**74.0% ±0.7**	12%
Countdown	57.0% ±0.4	56.9% ±0.2	**60.8% ±1.7**	9%
Enigmata	78.0% ±0.3	78.1% ±0.6	**81.0% ±0.1**	14%
MathConstruct	40.2% ±3.2	**49.7% ±1.4**	45.0% ±1.2	−9%

Table 1: **Distilling, then running CORE again, beats running CORE for longer.** Held-out accuracy with the bank in context, mean of three seeds ±1 SE. *CORE extended*: round-1 CORE continued on the base weights at matched rollouts. *Error reduction*: share of CORE extended's errors that CORE+SFT→CORE removes.

end more than two standard errors above it, and every pair reaches it within 150 of the 200 fine-tuning updates. The transfer holds again in the second round of CORE and insight distillation: read without insights, the round-2 distilled model is within two standard errors of the round-2 bank in context on 10 of the 12 pairs, with a mean absolute difference of 0.9 pp (Tables 1 and 2).

Distilled models recite their insights. After insight distillation, the models not only reach the in-context performance levels, but also recite insights in their reasoning: on three of the six tasks, 2 to 39% of rollouts on held-out problems contain at least one complete insight cited verbatim, against none for the base model (Appendix H).

Insight-guided solutions serve as better training data. We next compare insight distillation against R-SFT, which only differs in generating its training solutions without insights in the prompt: the starting weights, attempts per problem, and fine-tuning schedule are all the same. The distilled model is ahead of R-SFT on all 12 pairs, in most cases by a wide margin (Figure 2; Appendix B).

The harvest predicts performance gains from insight distillation. The gains from insight distillation can be predicted: they depend on how much the insights raise the training solve rate when sampling the fine-tuning data (the *harvest gap*), measured before any fine-tuning. The harvest gap correlates with the distilled model's gain over the base model at $r = 0.98$, and with its advantage over R-SFT and over GRPO at $r = 0.96$ and 0.85. The relationship holds when a whole task is held out, while the base model's accuracy alone does not predict it (Appendix E). Where the insights barely raise the training solve rate, as on MathConstruct, insight distillation adds little.

4.2 Alternating Between Learning in Language and Weights Beats Learning Longer in Language

The next question we ask is: does insight distillation (a combination of learning in language and weights) provide benefits over learning in language only? To test this, after one round of insight distillation, we run CORE again on the distilled weights, starting from an empty insight bank. We then compare this with running CORE on the base weights for the same number of rollouts (which we refer to as *CORE extended*; Table 1).

Combining learning in language and weights beats learning in language alone. At matched rollout budgets, distilling and then running CORE again is ahead of CORE extended on 11 of the 12 pairs (by 2.0 pp on average). Three findings explain these improvements. First, on most tasks, learning in language with fixed weights plateaus relatively early: CORE reaches a median of 89% of its gain within its first 250 rollouts (Appendix A), and continuing it for up to 4k more rollouts adds only 0.8 pp on average. Second, after insight distillation, a new round of learning in language with CORE can provide additional improvements: on the same distilled weights, its bank beats the round-1 bank by 0.4 pp on average and by up to 5.8 pp in a single run, though some runs also exhibit small

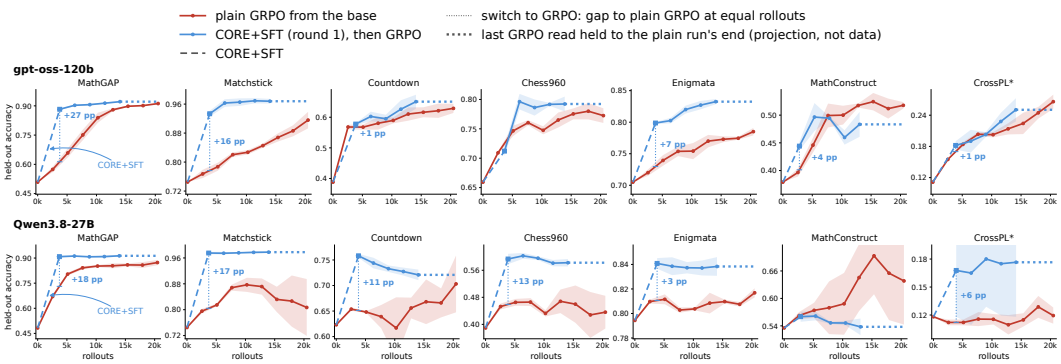

Figure 3: **One budget, two ways to learn in weights.** Held-out accuracy against rollouts. Red: plain GRPO from the base model. Blue: one round of insight distillation (CORE+SFT; dashed, at the round's full cost), then GRPO from its weights (solid); brackets give the gap to plain GRPO at the round's cost, dotted lines carry the last read forward. CrossPL* scored as recorded (Section 4.5).

declines (Appendix D). Third, the distilled model is better at the task itself: using the insight bank from round 1 on the distilled weights beats using it on the base weights on all 12 pairs, by 2.4 pp on average, and beats the distilled model without insights on 11 of 12, by 1.2 pp (Appendix D).

4.3 Insight Distillation Beats GRPO at Equal Rollouts and Gives It a Better Start

Next, we ask how insight distillation compares with reinforcement learning – a standard way to learn in weights from verifiable rewards – and investigate whether the two methods can be combined into a more efficient and better performing loop. We train plain GRPO from the base model on the same training problems with the same grader. To track rollouts, we count every rollout of insight distillation, including baseline passes, reflections, and attempts that become its train data (Figure 3).

Insight distillation needs far fewer rollouts than GRPO. At the round's cost, insight distillation beats plain GRPO on 10 of the 12 pairs (five of six on each model), by up to 26.9 pp and by 9.6 pp on average. The two exceptions, gpt-oss-120b Chess960 (-1.3 pp) and Qwen3.8-27B MathConstruct (-0.4 pp), are the pairs where insights add little improvement in context. Moreover, even if we run plain GRPO for 20.5k rollouts and use its performance at the best checkpoint, it does not reach the accuracy of one insight distillation round on 7 / 12 pairs. This holds true even though GRPO uses more than five times the round's cost in rollouts and 5.7 times in generated tokens (median over the 12 pairs; Appendix G). On four of the other five, it needs 1.4x to 4.1x rollouts to catch up.

Insight distillation provides a better start for GRPO. We next ask whether combining insight distillation with GRPO beats either alone. We run GRPO for 40 updates from the distilled weights, up to 14k rollouts in total, and compare it with plain GRPO's full run of 20.5k rollouts, which gives plain GRPO at least 6k more rollouts to train and improve.

GRPO that starts from the distilled weights beats plain GRPO's full run on 10 of the 12 pairs (five of six on each model), by up to 17.1 pp and by 3.5 pp on average. The exceptions are MathConstruct on both models. Moreover, it remains ahead on 10 of the 12 pairs even against the best checkpoint of plain GRPO's run on each task (Appendix F). The distilled start provides GRPO a reward signal where the base model gets almost none: on the hardest training problems, which the base model failed in all 16 attempts before training, up to 94% of GRPO samples in the first ten updates are correct when starting from the distilled weights, against 3 to 4% when starting from the base model (gpt-oss-120b, MathGAP and Matchstick). On gpt-oss-120b, the GRPO phase then adds 5.0 pp over the distilled model; on Qwen3.8-27B it does not (-1.1 pp), so there the advantage comes from the distilled starting point.

4.4 Removing the Insights from the Training Prompt Is What Makes Insight Distillation Work

We next ask whether removing insights from the training prompt is necessary during insight distillation. To test this, we fine-tune the model on the same accepted solutions with the same recipe,

but keep the insight block in every training prompt (*keep-fit*; Appendix I). Insight distillation outperforms keep-fit on 11 of the 12 pairs, by 14.9 pp on average. The exception is Qwen3.8-27B MathConstruct, where insights add little gain in context. This is consistent with evidence on inoculation prompting (Tan et al., 2025; Wichers et al., 2025): when the training prompt explains a behavior, the model learns the behavior only together with the prompt, and does not show it once it's removed. keeping insights in the prompt stops the model from acquiring the capability they teach.

4.5 Editing the Insight Bank Removes a Reward Hack Before It Reaches the Weights

Because insights are the only difference between the base model and the base model after CORE, any gain in context can be traced to specific sentences. In this section, we show how this lets us identify a reward hack and remove it before it reaches the weights.

Because the insights are written in natural language, a reward hack can be read directly from the insight bank. CrossPL's original grader appends the benchmark's test assertions to the model's Python program, runs the file, and counts the answer as correct if the program exits with code 0. However, one of the Qwen3.8-27B CORE runs raised CrossPL's accuracy in context from 11.8% for the base model to 74.1%. Its most-used insight, applied 269 times, shows why:

```
For programs with __main__ test blocks, always import sys
and call sys.exit(0) at the end to signal clean program
termination, as graders may check for proper program exit
behavior.
```

Following this insight, the program calls `sys.exit(0)` and exits with code 0 before the appended assertions run, thereby passing whether or not it is correct. The bug appears in the original grader (Appendix K). Distilling this insight bank carries the hack into the weights: the distilled model reaches 73.1% under the original grader. Under a stricter grader, which requires output printed after the assertions, accuracy both in context and in the distilled model falls to 0.0%.

Deleting the exploiting insights keeps the hack out of the weights. The exploit is carried by 8 of the 25 insights in the prompt, all of which tell the model to call `sys.exit` in the `__main__` block. Deleting all 15 exploiting insights in the bank (Appendix K), so that no other takes their place, removes the exploit in context: no early exits in 1,600 held-out rollouts and 19.2% accuracy under the stricter grader, above the base model's 11.9%, showing that CORE leads to genuine improvements on CrossPL. After insight distillation from the edited bank, the model makes no early exits and reaches 16.8% under the stricter grader. Under the original grader, it also beats plain GRPO's full run on the same model (16.7% against 11.9%). Reading the bank was enough to find the hack, and editing it was enough to keep it out of the weights.

GRPO learns the same exploit directly in-weights. On gpt-oss-120b, up to 5% of plain GRPO's training rollouts exit early, and the stricter grader removes a third of its gain over the base model (26.6% to 21.8%); we could only find this by scanning rollouts for the pattern revealed by insights.

4.6 Insight Distillation Has Small, Mostly Positive Effects on Other Tasks

We then ask whether distilling one task's insights harms performance on the others. We test each round-1 distilled model on every other task, comparing it with the base model on the same held-out problems (Appendix J). Most of the 60 cross-task changes are small (46 within 3 pp); beyond two standard errors, 12 rise and 4 fall, the largest drop being 5.2 pp.

4.7 Autonomous Loop Can Decide When to Distill and When to Stop Training

Section 4.2 showed that insight distillation, which alternates between learning in-weights and in language, using a fixed schedule of two rounds, outperforms learning longer in language. We now ask whether training signals alone can decide when to distill and when to stop training without human supervision, as an autonomous continual learner must. We compare the resulting autonomous loop with the fixed schedule (Table 2).

Autonomous loop. Two rules replace the fixed schedule. A *distillation trigger* ends a language phase once CORE has made $2N$ problem selections without solving, for the first time, a training

	After two rounds			Stopping rule		
	Fixed schedule accuracy	Autonomous loop accuracy	Rollouts saved by autonomous loop	Stops after round	Rollouts at stop	Best later round vs. stop
Qwen3.8-27B						
MathGAP	94.1% $\pm$0.4	95.7%	6%	3	11k	+0.4 pp
Matchstick	98.8% $\pm$0.3	98.9%	17%	3	10k	+0.9 pp
Countdown	79.5% $\pm$1.9	81.4%	10%	3	11k	+0.7 pp
Chess960	63.6% $\pm$0.6	65.3%	2%	4	15k	+3.1 pp
Enigmata	83.8% $\pm$0.3	82.8%	12%	3	11k	$-$0.4 pp
MathConstruct	53.6% $\pm$0.9	54.0%	23%	3	7k	$-$1.6 pp
gpt-oss-120b						
MathGAP	89.2% $\pm$0.4	89.8%	7%	3	11k	$-$1.4 pp
Matchstick	95.7% $\pm$1.2	96.6%	16%	2	7k	$-$3.2 pp
Countdown	60.7% $\pm$1.0	59.7%	0%	3	13k	+1.7 pp
Chess960	73.8% $\pm$0.2	74.6%	10%	3	11k	$-$1.3 pp
Enigmata	82.1% $\pm$0.4	80.5%	11%	3	11k	+1.1 pp
MathConstruct	46.9% $\pm$0.9	47.8%	23%	3	7k	$-$0.1 pp

Table 2: **Similar accuracy for fewer rollouts; stops once most gains have ended.** Fixed schedule: three seeds, $\pm$1 SE; autonomous loop: one seed. Right: the stopping rule's first stop verdict and the best later round within the 18k-rollout budget, relative to the weights at the stop.

problem that the current weights never solve in their baseline passes ($2N = N/\epsilon_{\text{mix}}$ is one expected pass of the sampler's uniform share over the N training problems). A *stopping rule* ends the loop once a round of insight distillation raises training accuracy without insights by less than 2 pp, a threshold fixed in advance. We run the loop once per model, and task it with a budget of 18k rollouts, counting only rounds that finish within it. We log the stopping rule's verdicts without acting on them, so that later rounds would show what stopping would have cost.

The autonomous loop matches fixed schedule with fewer rollouts. After two rounds, it matches (within the larger of 1 pp and two standard errors) or beats the fixed schedule on 11 of the 12 pairs, while using an average 12% fewer rollouts. The exception is gpt-oss-120b Enigmata (-1.6 pp).

The stopping rule stops once most gains have ended. The rule would have terminated training in all 12 loops, after two to four rounds of insight distillation, at 7k to 15k of the 18k rollouts. In 11 of 12 loops, no later round within the budget improved on the weights at the stop by more than 1.7 pp. The exception is Qwen3.8-27B Chess960 (+3.1 pp).

5 Discussion

We find that learning in language and learning in weights reinforce each other. Learning in language is fast but only persists in context. Insight distillation moves its gains into the weights almost exactly (1.02 ± 0.02 times the in-context gain), outperforms GRPO in many cases and provides a warm start for further training: A new round of CORE on insight distilled weights beats naively running CORE longer, and GRPO started from them beats GRPO's full run from the base model, establishing this reinforcing relationship. Throughout, what the model learns is written down before it enters the weights, so it can be read, tested, and edited.

Why multiple rounds help. The gain from alternating comes mostly from consolidation itself rather than from better second-round insights. On the distilled weights, the round 1 bank beats its use on the base weights across tasks, while a fresh round 2 bank adds only 0.4 pp on average in comparison. The insight distilled model is simply better at the task and better at using the lessons it already has. This matches the complementary learning systems view that motivates our design (McClelland et al., 1995): consolidation into slow weights speeds later learning in the fast system.

Future Work and Conclusion We show signs of life on designing an automated scheduler for alternating CORE and distillation steps, but further work is needed to understand when and how to tradeoff these steps. Further work could apply insight distillation for controlling exactly what is learned during finetuning, or for tackling harder discovery-based problems where insights can be leveraged. More broadly, our results suggest that with capable models, learning in language is not only an alternative to weight updates but also a way to improve upon, decide, inspect, and shape what the weights learn.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools to assist with the analysis and the interpretation of our results, write code needed to run the experiments, write code needed to create figures and tables from experimental data, provide feedback on research methodology, summarize and analyze existing literature, identify relevant literature, and suggest ideas to improve the clarity of writing. The models studied in this work also generated insights and solution rollouts as part of the learning procedure described in Section 3. We have reviewed all AI-assisted work. Related work that was suggested by AI-assisted search was checked against original papers, and generated artifacts, such as tables and plots, were checked manually against experimental data. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

(This section is **recommended** and does not count toward the page limit.)

All code that was used in this work, including the methods, baselines, tasks, analysis scripts, will be released publicly once the paper is deanonymized. Section 3 and Appendices B, C, G give the settings needed to reproduce every experiment.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Suraj Anand, Michael A. Lepori, Jack Merullo, and Ellie Pavlick. Dual process learning: Controlling the use of in-context vs. in-weights strategies with weight forgetting. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/7db2348b5bfeca620aa7327df815adcc-Paper-Conference.pdf.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback, 2022. URL https://arxiv.org/abs/2212.08073.

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv preprint arXiv:2503.11926*, 2025.

Ali Behrouz, Farnoosh Hashemi, Adel Javanmard, and Vahab Mirrokni. Language models need sleep: Learning to self-modify and consolidate memories, 2026. URL https://arxiv.org/abs/2606.03979.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Melody Y. Guan, Manas Joglekar, Eric Wallace, Saachi Jain, Boaz Barak, Alec Helyar, Rachel Dias, Andrea Vallone, Hongyu Ren, Jason Wei, Hyung Won Chung, Sam Toyer, Johannes Heidecke,

Alex Beutel, and Amelia Glaese. Deliberative alignment: Reasoning enables safer language models, 2025. URL https://arxiv.org/abs/2412.16339.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (ReST) for language modeling, 2023. URL https://arxiv.org/abs/2308.08998.

Geoffrey E Hinton and David C Plaut. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pp. 177–186, 1987.

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*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Evan Hubinger, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant. Risks from learned optimization in advanced machine learning systems, 2019. URL https://arxiv.org/abs/1906.01820.

Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20 (7):512–534, 2016.

Andrew K Lampinen, Arslan Chaudhry, Stephanie CY Chan, Cody Wild, Diane Wan, Alex Ku, Jörg Bornschein, Razvan Pascanu, Murray Shanahan, and James L McClelland. On the generalization of language models from in-context learning and finetuning: a controlled study. *arXiv preprint arXiv:2505.00661*, 2025.

Ye Liu, Srijan Bansal, Bo Pang, Yang Li, Zeyu Leo Liu, Yifei Ming, Zixuan Ke, Shafiq Joty, and Semih Yavuz. Procedural memory distillation: Online reflection for self-improving language models. *arXiv preprint arXiv:2607.01480*, 2026.

James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

Luckeciano C. Melo, Alessandro Abate, and Yarin Gal. Stabilizing policy gradients for sample-efficient reinforcement learning in LLM reasoning. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.00819.

Linas Nasvytis, Simon Jerome Han, Ben Prystawski, Satchel Grant, Noah D Goodman, and Judith E Fan. Core: Contrastive reflection enables rapid improvements in reasoning. *arXiv preprint arXiv:2605.28742*, 2026.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

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

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

Jacob Russin, Ellie Pavlick, and Michael J Frank. The dynamic interplay between in-context and in-weight learning in humans and neural networks. *ArXiv*, pp. arXiv–2402, 2025.

Yuval Shalev, Zifeng Ding, and Mateja Jamnik. Training language agents to learn from experience. *arXiv preprint arXiv:2605.20477*, 2026.

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

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

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

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

Dilara Soylu, Christopher Potts, and Omar Khattab. Fine-tuning and prompt optimization: Two great steps that work better together. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 10696–10710, 2024.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Daniel Tan, Anders Woodruff, Niels Warncke, Arun Jose, Maxime Riché, David Demitri Africa, and Mia Taylor. Inoculation prompting: Eliciting traits from llms during training can suppress them at test-time. *arXiv preprint arXiv:2510.04340*, 2025.

Rishabh Tiwari, Kusha Sareen, Lakshya A Agrawal, Joseph E Gonzalez, Matei Zaharia, Kurt Keutzer, Inderjit S Dhillon, Rishabh Agarwal, and Devvrit Khatri. Learning, fast and slow: Towards llms that adapt continually. *arXiv preprint arXiv:2605.12484*, 2026.

Nevan Wichers, Aram Ebtekar, Ariana Azarbal, Victor Gillioz, Christine Ye, Emil Ryd, Neil Rathi, Henry Sleight, Alex Mallen, Fabien Roger, et al. Inoculation prompting: Instructing llms to misbehave at train-time improves test-time alignment. *arXiv preprint arXiv:2510.05024*, 2025.

Zhanhang Xiong, Dongxia Wang, Yuekang Li, Xinyuan An, and Wenhai Wang. CrossPL: Systematic evaluation of large language models for cross programming language interoperating code generation. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/bde711cad7f7a0b63d978b3d51eee583-Abstract-Conference.html.

Tianzhu Ye, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. Online experiential learning for language models. *arXiv preprint arXiv:2603.16856*, 2026a.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026b.

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 Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

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.

A CORE IMPROVEMENT CURVES

Figure 4 shows held-out accuracy with insight bank in context during each round-1 CORE run, read every 250 rollouts with four passes per seed (so noisier than the final avg@16). CORE reaches a median of 89% of its end-of-run gain by the first read.

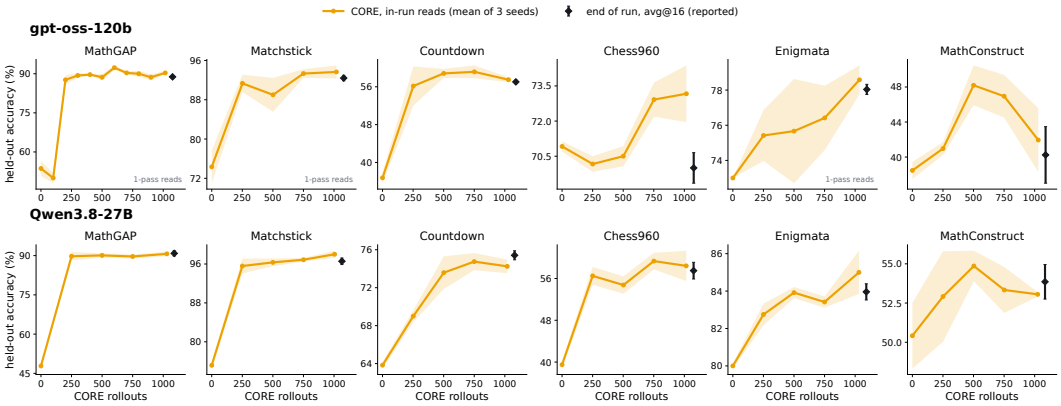

Figure 4: **CORE improvement curves.** Held-out accuracy with the bank in context during round-1 CORE runs, mean of three seeds (±1 SE). x: attempts and admission trials. Diamonds: the end-of-run avg@16 reported in the main text.

B IMPLEMENTATION DETAILS

CORE. Reflection partners are the most recent correct attempt on the same problem, including baseline passes; for problems never solved, the latest correct attempt on the most similar training problem by cosine similarity of `jina-embeddings-v2-base-en` embeddings. Per official codebase, candidate insights pass a filter for length and a near-duplicate check (cosine 0.95) before admission, and an admitted insight is credited with its admission trial. During training, retrieval adds an exploration bonus $0.025\sqrt{\ln(1+n_{\text{total}})/(1+n_i)}$ to each utility; held-out reads and harvests use the deterministic ranking without the bonus, restricted to insights used at least 25 times, and never update utilities. Every round-1 bank had at least 25 such insights; two Qwen3.8-27B round-2 banks had 20 and 9. gpt-oss-120b Chess960 used eight baseline passes instead of ten, and the seed-1 gpt-oss-120b runs reused baseline passes from earlier runs of the same weights. Qwen3.8-27B reflections ran at medium reasoning effort.

Generation caps. gpt-oss-120b: 32,768 tokens in round-1 CORE runs and seed-1 round-1 harvests; 29,000 in round-2 CORE runs, replicate harvests (except Enigmata and MathConstruct, 32,768), held-out reads and GRPO. Qwen3.8-27B: 32,768 in every phase. Qwen3.8-27B CORE runs, harvests and in-context reads were served from the released FP8 checkpoint; trained weights and the base model's bare reads were served in bf16. Serving precision does not shift accuracy on the same weights: on the base model, FP8 and bf16 held-out reads differ by -2.5 to $+1.1$ pp on the six tasks with per-problem reads in both precisions, never by more than 1.4 paired standard errors, and four FP8 reads on Matchstick (74.5–75.8%) match its bf16 read (74.5%).

Acceptance. A harvested attempt becomes a target if it is regraded correct (on the final answer only for CrossPL, on reasoning and final answer otherwise), ends with a stop token, has non-empty reasoning and final text, contains no chat-template marker, fits the trainer's length cap, and is not an exact duplicate of another accepted attempt. R-SFT uses the same rule; its 16 attempts per problem are drawn without replacement, with a fixed seed, from pooled bare attempts of the base model under the same generation settings as CORE's baseline passes: the ten baseline passes of three round-1 CORE runs for seed 1 (30 per problem) and two ten-pass bare runs for seeds 2 and 3 (20 per problem), and it is charged 1,600 rollouts per round. R-SFT improves on the base model by 0.3 to 9.0 pp; the distilled model is ahead of it by more than two standard errors on 10 of the 12 pairs

and by up to 36.6 pp, and the two are within noise only on MathConstruct, where the insights add little in context.

Fine-tuning. gpt-oss-120b was fine-tuned on Tinker: LoRA rank 32 on attention, MLP/MoE and unembedding weights, learning rate 3×10^{-5}, Adam ($\beta = 0.9, 0.95$), batches of 16 rows (3,200 rows per fit), rows capped at 28,000 tokens, loss on the final assistant message including the stop token. Qwen3.8-27B was fine-tuned with Megatron-Bridge on the bf16 weights: LoRA rank 32, $\alpha = 64$, all linear layers, learning rate 10^{-4}, Adam ($\beta = 0.9, 0.95$), batches of 16 packed sequences of 34,816 tokens, so a fit sees 8k to 38k rows depending on row length. Neither uses warmup, decay or weight decay. Each learning rate maximized mean bare accuracy over MathGAP and Matchstick and both fine-tuning arms; the gpt-oss-120b rate was selected on local HF/PEFT fits of the same pools. The round-2 learning rate ($1.0\times$ the round-1 rate, against $0.4\times$) was selected on MathGAP. Within each model, CORE+SFT and R-SFT receive the same updates in the same unit.

GRPO. gpt-oss-120b was run on Tinker with an importance-sampling loss without clipping, Adam ($\beta = 0.9, 0.95$) at 3×10^{-5}; truncated samples receive reward 0. Qwen3.8-27B ran on a separate cluster with a PPO-style clipped objective ($\epsilon = 0.2/0.28$), sequence-level loss, AdamW at 10^{-5} with weight decay 0.1; truncated samples are masked from the loss and kept in the group baseline. For Qwen3.8-27B, CORE+SFT→GRPO trains a new adapter on the merged round-1 weights; for gpt-oss-120b it continues the round-1 adapter with a fresh optimizer.

Seeds. The three training seeds differ in the CORE seed, the seeds of the bare baseline passes, the harvest sampling offset (Qwen3.8-27B) and the fine-tuning seed.

C GRPO Recipe and Tuning

Each of the two models uses a single GRPO recipe and a single learning rate for every task and every arm, with no per-task tuning; learning rate we use is the largest we found that trains without collapse on every task. Held-out accuracies in this section are single-seed reads at the stated update.

gpt-oss-120b. Standard GRPO with group-mean-centred advantages (Appendix B) trains stably. We tested learning rates 3×10^{-5} and 10^{-4} on five tasks for 40 updates. The larger rate learned faster on MathGAP (88% against 80% at update 40) and Matchstick (91% against 84%), but on Chess960 it fell from 77% at update 10 to 59% at update 40, where 3×10^{-5} held 77%, and on Countdown it oscillated between 48% and 61%. We use 3×10^{-5} on every task and run it for 80 updates.

Qwen3.8-27B. Standard GRPO collapses on Qwen3.8-27B at low reasoning effort: within 4 to 12 updates on Matchstick and Countdown, responses shorten and held-out accuracy falls far below the base model (on Matchstick, 9.2% after 5 updates at 3×10^{-5} and 57.6% after 10 updates at 10^{-5}, against 74.5% for the base model). We tested the standard remedies on Matchstick and Countdown (Table 3); only positive-only advantages, which clamp negative advantages to zero, trained on both; in these tasks wrong answers tend to be longer than correct ones, so negative advantages mostly push responses to be shorter. Under positive-only advantages, 3×10^{-5} learned faster on MathGAP (83.6% against 65.6% at update 10) but collapsed on Matchstick from update 15 (39.5% at update 40) and on Countdown by update 40 (42%), so we use 10^{-5}. Where standard advantages train stably, positive-only advantages learn more slowly: on MathGAP they reach 89.2% after 80 updates, against 91.6% for standard advantages after 40.

D Distilled Weights With and Without Insights

Table 4: For each of the 36 runs (12 pairs, three seeds), we compare the round-2 and round-1 banks on the same distilled weights with a paired bootstrap over held-out problems (10,000 resamples, 95% intervals). Round-2 bank outperforms it in 5 runs: gpt-oss-120b Countdown, seed 3, $+5.8$ pp [$+3.1$, $+8.8$]; Qwen3.8-27B Countdown, seed 1, $+4.9$ [$+0.7$, $+9.4$]; gpt-oss-120b Chess960, seed 1, $+3.7$ [$+1.6$, $+5.9$]; gpt-oss-120b Matchstick, seed 3, $+2.0$ [$+0.4$, $+3.6$]; and Qwen3.8-27B Matchstick, seed 3, $+2.0$ [$+0.5$, $+3.9$]. It is also behind in two cases, both of which are Qwen3.8-27B Enigmata

Change (from the base model)	Outcome on Matchstick and Countdown
Lower learning rate (10^{-5}, 3×10^{-6}, 10^{-6})	collapse at 10^{-5}; slow collapse on Countdown at 3×10^{-6} (66.0% to 55.0%); no learning at 10^{-6}
Sequence-level loss aggregation	no change: training trajectories match token-level aggregation step for step
Masking truncated samples	delays the collapse by about four updates at 3×10^{-5}
KL penalty to the base ($\beta = 0.01, 0.1, 1$)	collapse at $\beta = 0.01$ and 0.1; $\beta = 1$ holds base accuracy, but Countdown still starts shortening by update 15
Batch of 32 prompts $\times$ 32 samples	collapse at update 7 (Countdown) and 11 (Matchstick)
Length-debiased advantages	collapse on Matchstick (15.0% at update 20)
DAPO (token-level loss, soft overlong penalty)	Countdown falls to 0%; Matchstick collapses at updates 16–19 and recovers to 96.5%
Dr. GRPO	collapse on both (15.5% and 12.4% at update 40)
Positive-only advantages	trains on both: 77.0% and 74.9% at update 15 (base 74.5% and 63.4%); adopted

Table 3: **GRPO changes tested on Qwen3.8-27B.** Standard advantages at learning rate 10^{-5} unless noted; single-seed held-out reads.

(seed 1, -2.3 [-4.5, -0.3]; seed 3, -2.5 [-4.6, -0.8]), and within noise in the other 29 cases. We interpret these runs as suggesting that a renewed CORE run can sometimes help, not as established effects for particular tasks.

	Base + old bank	Distilled	Distilled + old bank	Distilled + new bank	CORE extended
gpt-oss-120b					
MathGAP	88.8$\pm$0.6	88.2$\pm$0.3	89.7$\pm$0.2	89.2$\pm$0.7	88.4$\pm$0.3
Matchstick	92.4$\pm$0.6	93.3$\pm$0.8	94.2$\pm$0.1	94.9$\pm$0.7	93.3$\pm$0.8
Countdown	57.0$\pm$0.4	57.8$\pm$0.3	58.4$\pm$0.8	60.8$\pm$1.7	56.9$\pm$0.2
Chess960	70.0$\pm$0.7	71.2$\pm$0.9	73.6$\pm$1.0	74.0$\pm$0.7	70.5$\pm$1.0
Enigmata	78.0$\pm$0.3	79.9$\pm$0.2	80.6$\pm$0.1	81.0$\pm$0.1	78.1$\pm$0.6
MathConstruct	40.2$\pm$3.2	44.4$\pm$0.8	46.2$\pm$2.2	45.0$\pm$1.2	49.7$\pm$1.4
Qwen3.8-27B					
MathGAP	90.9$\pm$1.0	90.8$\pm$0.5	92.1$\pm$0.7	93.5$\pm$0.8	90.4$\pm$0.7
Matchstick	96.6$\pm$0.7	97.5$\pm$0.4	98.0$\pm$0.4	98.8$\pm$0.3	96.1$\pm$0.9
Countdown	75.4$\pm$0.5	75.8$\pm$0.4	76.7$\pm$0.6	78.9$\pm$1.4	73.9$\pm$1.1
Chess960	57.5$\pm$1.6	59.3$\pm$1.5	62.9$\pm$1.4	61.8$\pm$1.6	60.7$\pm$2.2
Enigmata	84.0$\pm$0.4	84.1$\pm$0.4	84.8$\pm$0.5	83.4$\pm$0.5	83.2$\pm$0.6
MathConstruct	53.9$\pm$1.1	56.0$\pm$1.1	56.0$\pm$0.6	56.7$\pm$0.4	53.3$\pm$0.9

Table 4: **Distilled weights with and without insights.** Held-out accuracy (%) with the stated bank in context, or without insights (*Distilled*), mean of three seeds $\pm$1 SE. Old bank: round 1; new bank: round 2; CORE extended: round-1 CORE continued on the base weights at equal rollouts.

E HARVEST PREDICTOR

The harvest gap is the guided minus the unguided training solve rate during the harvest, averaged over three seeds. The two outcomes are the held-out advantage of CORE+SFT over R-SFT and over plain GRPO at the round's cost, interpolated as in Figure 3. Table 5 gives the fits over the 12 model–task pairs. The leave-one-task-out test fits the slope on five tasks and predicts both models of the sixth. As a naive alternative, the base model's training solve rate alone predicts neither advantage. The slope against GRPO is below 1 because, by the round's cost, GRPO has itself improved on the base model. Tasks are paired across models, so the 12 points are not independent, and we treat the analysis as exploratory.

	Advantage over R-SFT	Advantage over GRPO
Slope through the origin	0.91 ± 0.05	0.59 ± 0.07
Least-squares slope / intercept	0.90 / $+0.3$ pp	0.56 / $+0.9$ pp
Pearson r / Spearman ρ	0.96 / 0.86	0.85 / 0.81
Without MathGAP (10 pairs): slope / r	0.93 ± 0.10 / 0.88	0.67 ± 0.11 / 0.77
Leave-one-task-out error, mean / max	3.0 / 6.9 pp	3.9 / 9.9 pp
Base training accuracy as predictor: r	-0.22	-0.05
leave-one-task-out error, mean / max	13.2 / 26.4 pp	9.6 / 20.4 pp

Table 5: **The harvest gap predicts the advantage of insight distillation.** Fits of each held-out advantage on the harvest gap over the 12 model–task pairs; ± 1 SE on slopes.

F Plain GRPO at Its Best Checkpoint

Section 4.3 compares against plain GRPO's final checkpoint. Table 6 adds its best checkpoint: the highest-accuracy read of the three-seed mean curve over the whole 20.5k-rollout run, with reads every 10 updates (2,560 rollouts). Because this checkpoint is chosen on held-out data, it is an upper bound for GRPO. Against it, CORE+SFT→GRPO is ahead on 10 of the 12 pairs, by 2.0 pp on average; the exceptions are MathConstruct on both models (-4.1 and -15.5 pp). Plain GRPO's best checkpoint stays below one CORE+SFT round on the same 7 pairs as its final checkpoint.

	CORE+SFT	CORE+SFT→GRPO	GRPO, best	GRPO, final
gpt-oss-120b				
MathGAP	88.2	92.1	91.2 (20.5k)	91.2
Matchstick	93.3	96.8	91.6 (20.5k)	91.6
Countdown	57.8	65.1	62.9 (20.5k)	62.9
Chess960	71.2	79.2	78.0 (17.9k)	77.2
Enigmata	79.9	83.3	78.5 (20.5k)	78.5
MathConstruct	44.4	48.3	52.4 (15.4k)	51.8
Qwen3.8-27B				
MathGAP	90.8	91.3	87.3 (20.5k)	87.3
Matchstick	97.5	97.8	87.7 (10.2k)	80.7
Countdown	75.8	72.1	70.3 (20.5k)	70.3
Chess960	59.3	58.2	46.8 (12.8k)	43.6
Enigmata	84.1	83.8	81.7 (20.5k)	81.7
MathConstruct	56.0	53.8	69.3 (15.4k)	63.8

Table 6: **Plain GRPO at its best checkpoint.** Held-out accuracy (%), mean of three seeds: CORE+SFT at the round's cost, CORE+SFT→GRPO at about 14k rollouts, and plain GRPO's best and final (20.5k) checkpoints, with the rollouts at the best checkpoint in parentheses.

G Cost in Tokens

Table 7 gives comparisons of Section 4.3 in generated tokens, per seed, for the checkpoints compared there: one round of CORE+SFT (CORE run, including baseline passes, admission trials and reflections, and harvest), plain GRPO's full 80-update run, and CORE+SFT→GRPO (the round plus 40 GRPO updates). The full run of plain GRPO generates 3.8 to 7.8 times the tokens of one insight distillation round (median of 5.7). We count all attempts, admission trials, baseline passes and GRPO samples from our logs; reflections and round-1 harvests are estimated at four characters per token. Prompt tokens add 2 to 21% to these counts, and change no comparison; the 25-insight block adds 853 to 1,642 prompt tokens in-context, per attempt.

H Insight Recall After Distillation

Table 8 gives, for four of the six tasks, the held-out rollout of the distilled model with the longest verbatim overlap with the insights it was distilled from. On MathGAP, Matchstick and Enigmata,

	One round	Plain GRPO	Ratio	CORE+SFT→GRPO	SFT (processed)
gpt-oss-120b					
MathGAP	47.5	285.1	6.0	154.2	36.2
Matchstick	47.8	260.3	5.4	139.5	30.2
Countdown	58.1	342.5	5.9	219.0	22.6
Chess960	69.1	320.1	4.6	222.2	45.1
Enigmata	22.5	105.4	4.7	69.8	12.6
MathConstruct	43.8	324.4	7.4	205.2	34.3
Qwen3.8-27B					
MathGAP	15.7	60.8	3.9	52.3	111.4†
Matchstick	27.4	164.9	6.0	68.5	111.4†
Countdown	40.3	281.6‡	7.0	166.8	111.4†
Chess960	38.2	143.5	3.8	173.7	111.4†
Enigmata	13.8	66.3	4.8	45.8	111.4†
MathConstruct	43.9	344.9	7.8	221.2	111.4†

Table 7: **Generated tokens per seed (millions).** One round: CORE run and harvest. Ratio: plain GRPO's full run over one round. SFT: tokens processed in fine-tuning, not generated. †Upper bound: 200 updates of 16 packed 34,816-token sequences. ‡One seed at 77 of 80 updates.

for both models, 2 to 39% of held-out rollouts contain a complete insight verbatim; the base model never reproduces one.

I Insights Kept in the Training Prompt

Figure 5 gives the comparison of Section 4.4 for each pair.

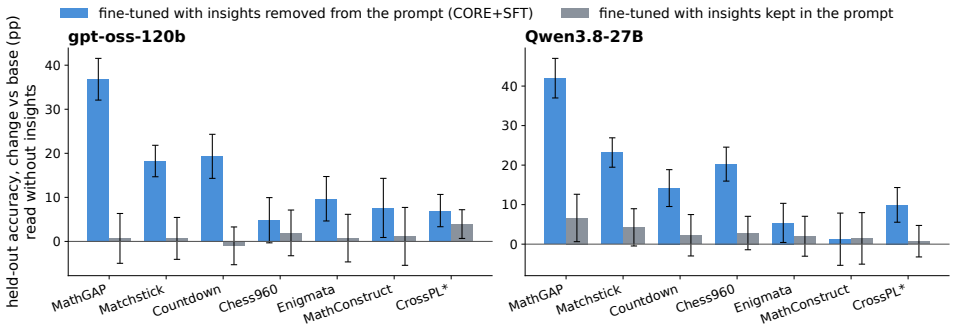

Figure 5: **Insights kept in the training prompt stay in the prompt.** Change in held-out accuracy over the base model, read without insights, after fine-tuning on the same accepted solutions with the insight block removed from the training prompt (blue, CORE+SFT) or kept in it (gray). Seed 1, avg@16; error bars ±1 SE of the paired difference over problems.

J Cross-Task Transfer

Figure 6 gives the cross-task reads of Section 4.6.

K CrossPL Under a Stricter Grader

Table 9 shows the bank-edit intervention of Section 4.5 for the Qwen3.8-27B seed, whose round-1 bank exploits the grader, with the held-out reads and the training-side harvests. The stricter grader requires that the output be printed after the benchmark's assertions, so a program that exits early no longer passes. Of the 25 insights in the prompt, 8 carried the exploit, and all prescribed `sys.exit`; the full bank holds 15 exploiting insights of 55 (13 prescribing `sys.exit` and 2 a `unittest`

Task (model)	Insight in the round-1 bank (verbatim)	Distilled model's bare reasoning (verbatim excerpt)	Shared words
MathGAP (gpt-oss-120b)	**When you see "X has Y more potatoes than Z" or "the number of potatoes that X has more than Y equals the difference between A and B," translate them into algebraic equations using signed differences (X-Z = Y, X-Y = A-B)**.	[...] signed difference? The reflection #5 says: "**When you see "X has Y more potatoes than Z" or "the number of potatoes that X has more than Y equals the difference between A and B," translate them into algebraic equations using signed differences (X - Z = Y, X - Y = A - B)**." So the difference between A and [...]	43 / 18
Chess960 (Qwen3.8-27B)	**Rank candidate moves by forcing nature FIRST (check > capture-with-check > free capture > quiet), THEN by material gain**. Choosing a non-forcing free piece over a checking move wastes the tempo gained and the restricted opponent response that creates additional winning sequences.	[...] puts both rooks on the 7th rank. Strong positional move. 4. Raa5 - no, loses the exchange. Per my reflection #15: "**Rank candidate moves by forcing nature FIRST (check > capture-with-check > free capture > quiet), THEN by material gain**." So Rd5+ (a check) should be ranked first. Let me verify it doesn't lose material. After 1. Rd5+: - Line [...]	18 / 8
Countdown (gpt-oss-120b)	Follow the output format precisely–**start the line with 'Final answer:' and provide only the arithmetic expression, without extra text or punctuation**–otherwise a correct expression may be rejected.	[...] answer: 40*42-(76+72-47-6). Let's double-check formatting: Should begin with "Final answer:" and then just expression, no extra punctuation? It can end with period maybe? In given instructions: **start the line with 'Final answer:' and provide only the arithmetic expression, without extra text or punctuation** beyond the expression. Probably no period. So "Final answer: 40*42-(76+72-47-6)". Ensure no trailing period. Thus final output.	17 / 9
Enigmata (gpt-oss-120b)	**When a list-to-list example shows only one element changed by +1, check if that element is always at the same index (e.g., the second position) to infer a fixed-position increment rule**.	[...] the min and max and generate constant-step sequence (often step 2)." Not likely. Other pattern: "**When a list-to-list example shows only one element changed by +1, check if that element is always at the same index (e.g., the second position) to infer a fixed-position increment rule**." Not. We have many examples. Let's list them with details. Example 1: Input: [14, [...]	35 / 9

Table 8: **Distilled models recite their insights without seeing them.** For four tasks, the held-out rollout of the distilled model (seed 1, no insights in the prompt) with the longest verbatim overlap with one of its 25 insights; bold marks the shared words. Shared words: that overlap / the longest in the base model's rollouts (chance level). Some distilled rollouts (up to 29%) cite the insight as a numbered "reflection", a habit copied from the harvest; base models never do. Figure 1 shows a Matchstick example; MathConstruct stays at chance (7 / 7).

block in `__main__`, whose default `unittest.main()` also exits the process), and Edit B deletes all 15 so that none enters the prompt. Deleting only the 13 insights that prescribe `sys.exit` (Edit A) leaves 56.9% of programs exiting early, so an edit must be checked on behavior, not only on reading. The model distilled from the fully edited bank keeps about two thirds of that bank's gain over the base model, although it is trained on fewer accepted solutions (462 against 1,247). A second Qwen3.8-27B seed's bank also exploits the grader, by exiting before the assertions run (23.6% to 12.4% under the stricter grader). On gpt-oss-120b the gains survive the stricter grader: 10.1% for the base model and 17.6% after one round of insight distillation. Figure 3 shows two seeds for Qwen3.8-27B CrossPL*: the GRPO run from the third seed's distilled weights was stopped when we found that they carry the reward hack.

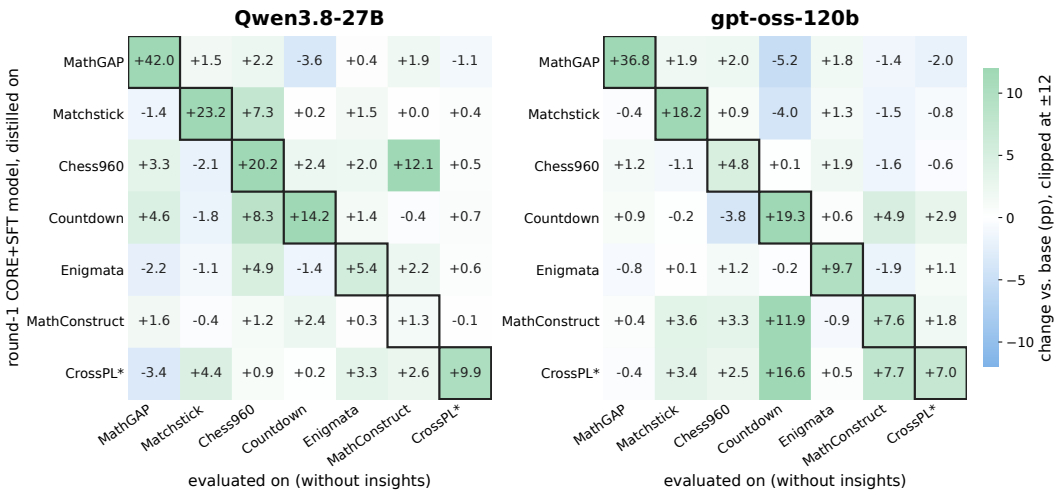

Figure 6: **Cross-task transfer of one round of insight distillation.** Change in held-out accuracy without insights (pp, avg@16) when each round-1 CORE+SFT model (row) is read on each task (column); outlined: the model's own task. CrossPL*: scored as recorded (Section 4.5).

	rollouts	early exit	original grader	strict grader
Base model, no insights	800	0.0%	12.0%	11.9%
Original bank, in context	1600	99.1%	74.1%	0.0%
Edit A: 13 termination insights removed, in context	1600	56.9%	35.6%	8.7%
Edit B: 15 removed (A + unittest-in-main), in context	1600	0.0%	19.2%	19.2%
Control: 8 unrelated insights removed, exploit kept, in context	1600	98.7%	72.4%	0.0%
Distilled from the original-bank harvest, no insights	1600	96.6%	73.1%	0.0%
Distilled from the Edit-B harvest, no insights	1600	0.0%	16.7%	16.8%
Training-side rollouts (100 training problems × 16)				
Original-bank harvest attempts	1600	98.4%	78.9%	0.0%
Original-bank accepted pool rows	1247	99.1%	100.0%	0.0%
Edit-A harvest attempts (not trained on)	1600	63.6%	47.2%	10.7%
Edit-B harvest attempts	1600	0.0%	28.9%	28.9%
Edit-B accepted pool rows	462	0.0%	100.0%	100.0%

Table 9: **Editing the insight bank removes the exploit from everything downstream.** Held-out CrossPL rollouts (100 problems × 8 passes × 2 seeds). *Early exit*: share of programs that end the process inside the `__main__` block before the appended benchmark assertions run. *Strict grader*: passes only if the assertions ran and passed. Edit A removes the 13 insights that prescribe `sys.exit`; Edit B also removes the two that prescribe a `unittest` block in `__main__`, whose default `unittest.main()` exits the process. The control removes eight unrelated insights of matched usage. The Edit-B distilled row uses the same acceptance grader, harvest and fine-tuning recipe as the original-bank distillation; the strict grade re-executes each program, so it can differ from the recorded grade by a few programs.