CORE: CONTRASTIVE REFLECTION ENABLES RAPID IMPROVEMENTS IN REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models can use verifiable rewards to improve at a wide variety of reasoning tasks. However, both parametric (e.g. RLVR) and non-parametric (e.g. prompt optimization) approaches to doing so typically require hundreds of training samples and thousands of model rollouts, making them expensive in the best case and intractable in the worst. To address this challenge, we introduce Contrastive Reflection (CORE), a non-parametric learning algorithm that compares past reasoning traces to generate *insights*: short natural-language descriptions of reasoning strategies and constraints that capture differences between successful and unsuccessful problem attempts. Across four reasoning tasks, we demonstrate that CORE enables more rapid improvement than both parametric (GRPO) and non-parametric (GEPA, episodic RAG, and MemRL) methods, while using fewer rollouts. Under fixed rollout budgets with as few as five training samples, CORE achieves the strongest performance in most task–data regimes. Finally, we highlight how CORE is substantially more context-efficient than non-parametric baselines, requiring fewer prompt tokens while storing learned knowledge as human-interpretable natural-language insights. Our results therefore suggest that distilling contrasts between successful and unsuccessful reasoning traces into abstract and useful insights can provide a more efficient and interpretable route to model self-improvement than weight updates, prompt optimization, or direct reuse of stored reasoning traces.

1 INTRODUCTION

Language models can learn from verifiable rewards, but often require large amounts of data and compute to do so. For example, parametric methods such as GRPO can require hundreds of thousands of rollouts for a model to make meaningful progress on a given task (Guo et al., 2025), while non-parametric methods such as GEPA use fewer rollouts but still rely on hundreds of training and validation samples to make comparable gains (Agrawal et al., 2025). Humans, by contrast, can often improve substantially at a new task with only a handful of practice problems and learning trials (Nam & McClelland, 2024). What accounts for this difference, and how might we enable language models to learn from verifiable rewards with more human-like efficiency?

Here, we consider *insight discovery* as one potential answer to these questions. A rich body of work in cognitive psychology suggests that rapid learning in humans depends partly on the ability to revisit past successes and failures in order to discover more abstract, explicit, and concise principles that explain their difference (Cushman, 2020; Schwartz & Bransford, 1998; Alfieri et al., 2013). Comparing what has worked to what has not is crucial, as is estimating how useful these learned insights are: people often acquire more general and reusable insights when contrasting past experiences rather than reflecting upon them in isolation (Schwartz & Bransford, 1998; Alfieri et al., 2013), and people also often apply insights selectively when solving new problems based on what is currently relevant or was previously useful (Lisman & Grace, 2005; Adcock et al., 2006). Although prior work has shown that language models can improve beyond scalar rewards alone with the use of verbal reflections or text-based feedback (Shinn et al., 2023; Song et al., 2026; Shi et al., 2026), these contrastive and utility-based mechanisms that can consolidate prior experience into more abstract, reusable knowledge are largely absent from existing approaches to learning from verifiable rewards.

To that end, we introduce Contrastive Reflection (CORE), a non-parametric learning algorithm that enables a frozen language model to learn from verifiable rewards with both greater sample and

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.

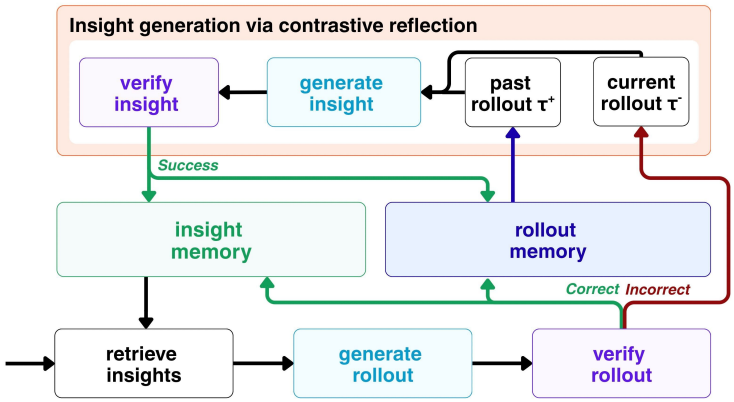


Figure 1: Illustration of the CORE algorithm: A model retrieves relevant insights, then generates a reasoning trace and answer to a question. If the model does not answer correctly, we generate new insights by contrasting the failed reasoning trace against a similar problem that the model answered correctly. Insights that lead the model to solve the problem correctly are added to the memory store.

rollout efficiency than existing methods. CORE works by building two external memory stores: one that stores generated insights ('insight memory'), and one that stores past rollouts ('rollout memory'). With every failed problem attempt during training, CORE prompts the language model to contrastively reflect by comparing pairs of its own successful and unsuccessful past rollouts in order to generate insights about the strategies and constraints that might capture differences between them. These insights are then selectively retrieved and placed in-context when the model encounters future problems on the basis of both semantic similarity and utility estimates that are updated with each new associated success and failure. CORE therefore differs from existing approaches to learning from verifiable rewards in what it stores: transparent, natural-language insights paired with utility estimates, rather than weight updates, individual rollouts, or a single global prompt.

Our contributions are as follows:

1. We introduce CORE, a non-parametric learning algorithm that enables language models to learn from verifiable rewards by generating and accumulating insights.
2. Across a range of reasoning tasks, we demonstrate that CORE can outperform competing parametric and non-parametric baselines with greater rollout, sample, and context efficiency.
3. We demonstrate that CORE generates *interpretable* learning artifacts that reduce the risk of introducing unwanted behaviors from opaque parameter updates.
4. We identify the components for generating useful insights through ablations that demonstrate how contrastive reflection and utility-aware retrieval each contribute to performance.

2 Related Work

Learning from Verifiable Rewards. Existing methods for learning from verifiable rewards can be classified as parametric or non-parametric. Parametric methods update model weights, for example by training on idealized rationales (STaR (Zelikman et al., 2022)) or by rewarding successful attempts via automatic verifiers (GRPO (Cobbe et al., 2021; Shao et al., 2024; Guo et al., 2025)). Non-parametric methods keep the model frozen and instead improve its context, for example by refining a single task prompt with task metrics (MIPRO) or textual feedback (GEPA) (Opsahl-Ong et al., 2024; Agrawal et al., 2025). Both types of approaches typically encode what is learned in forms that are either hard to interpret or hard to reuse selectively. In contrast to these methods, CORE keeps the model frozen but stores what is learned as a set of interpretable natural-language insights.

External Memory Systems for Language Models. Current external memory systems for language models typically vary along two dimensions that affect sample efficiency: what is stored, and how items are selected for retrieval. Stored items range from raw experience – full reasoning traces and

verbal reflections on task feedback (Shinn et al., 2023) – to increasingly abstracted forms: executable programs (Wang et al., 2023), distilled behavior descriptions (Didolkar et al., 2025; Ouyang et al., 2025), and human-authored procedural guidance such as `AGENTS.md` (Xu & Yan, 2026). Retrieval is typically based on semantic similarity, with MemRL extending this by learning utility scores over episodic memories (Zhang et al., 2026b). However, recent work shows that continuously consolidating trajectories into textual memory can be unstable, sometimes even degrading performance below no-memory baselines. This motivates memory systems that curate *which* raw episodes are used for consolidation and *when* consolidation occurs, rather than updating textual memory after every interaction (Zhang et al., 2026a). CORE differs from these approaches along both dimensions. It stores natural-language insights derived from curated contrasts between failed attempts and successful attempts on semantically similar problems, triggers this abstraction step only after failures, and admits candidate insights only when they improve performance on the originating problem. It retrieves these insights using both semantic similarity and each insight's measured utility on related problems.

Learning Efficiency. Recent methods for language-model self-improvement still require substantial compute and data for training, trading off between two dimensions of learning efficiency: *sample efficiency*, the number of distinct training problems needed for learning, and *rollout efficiency*, the number of attempts needed for those problems. Among parametric methods, Reinforcement Learning from Text Feedback (RLTF) adds natural-language critiques to scalar rewards to improve rollout efficiency, but its largest reasoning setting still trains on 19,800 Reasoning Gym examples (Song et al., 2026). Experiential Reinforcement Learning (ERL) improves rollout efficiency by adding a gated reflection-and-retry stage for sparse and delayed feedback, but still uses 10,000 procedurally sampled training instances for FrozenLake and Sokoban (Shi et al., 2026). Process-supervision methods can improve rollout efficiency by providing denser step-level feedback, but require expensive step-level labels or reliable process reward models (Lightman et al., 2023). Among non-parametric methods, GEPA improves rollout efficiency over GRPO on some tasks, but still uses up to 7,000 rollouts per task (Agrawal et al., 2025). CORE addresses both efficiency dimensions: by extracting insights from each scored attempt and later assigning them credit for improved or diminished performance, it can match or exceed these baselines using fewer training problems and fewer rollouts.

3 CORE: CONTRASTIVE REFLECTION

CORE is a non-parametric learning algorithm that allows a frozen language model to improve using explicit, contrastive, and utility-aware reflection. The central learned object in CORE is an *insight*: a short natural-language description of a general reasoning strategy or constraint that can help with future problems. Insights are not summaries of prior rollouts; instead, they are better viewed as credit-assignment hypotheses about what distinguishes successful rollouts from unsuccessful ones. CORE produces insights by contrastively reflecting about pairs of past rollouts, tests them by using verifier feedback, stores them for future use only when they improve model performance, and retrieves them on the basis of their continuously updated utility estimates. Figure 1 illustrates a single training step for this algorithm, and pseudocode can be found in Appendix A. Beyond the frozen language model M itself, CORE's implementation relies on four components: an external memory store, failure-biased problem sampling, insight retrieval, and contrastive reflection.

Problem setting. Let $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{eval}}$ denote training and held-out evaluation problem sets for a given task. We consider tasks where each problem $q \in \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{eval}}$ is associated with an existing verifier V_q that maps a candidate answer to a binary reward $r \in \{0, 1\}$. The solver is a language model M that remains frozen throughout. For each training problem q we maintain a baseline success-rate estimate b_q which captures the expected performance of M on q without learning updates. b_q is initialized from n_{base} independent samples and updated when additional observations are collected absent learning updates. This allows the baseline-relative utility of any learning update to be defined as $U(q) = r - b_q$. This normalization gives positive evidence only when an update-conditioned attempt improves over the baseline for that problem, preventing inflation from easy problems.

External memory store. CORE maintains two external memory stores. The 'rollout memory' $\mathcal{R}$ stores correctly solved past rollouts. A rollout $\tau = (q, I, y, r)$ contains a problem q, the retrieved set of insights I, the model output y, and the verifier reward r. Each rollout is indexed by an embedding of its problem. Rollout memory is used to find semantically similar past problems and construct pairs

of rollouts for contrastive reflection. The insight memory $\mathcal{I}$ stores short natural-language insights together with empirical evidence about where each insight has been useful. For each problem p on which insight i has been applied, CORE stores the number of applications $N_{p,i}$ and the empirical mean baseline-relative utility $\bar{U}_{p,i}$ for that insight–problem pair.

Failure-biased problem sampling. Because CORE only generates new insights after failures, we bias our problem sampling towards problems that the model solves less reliably. For training problem q_i, let n_i^{base} be the number of baseline samples used to estimate b_i, n_i^{epi} be the number of training attempts on q_i, and c_i^{epi} be the number of correct attempts. CORE estimates current accuracy as

$$a_i = \frac{n_i^{\text{base}} b_i + c_i^{\text{epi}}}{n_i^{\text{base}} + n_i^{\text{epi}}},$$

and assigns sampling weight $w_i = 1 - a_i$ to q_i. To maintain coverage of the training set, CORE mixes this failure-biased distribution with uniform sampling. If $N = |\mathcal{D}_{\text{train}}|$, then:

$$p_i = (1 - \epsilon_{\text{mix}}) \frac{w_i}{\sum_{j=1}^{N} w_j} + \epsilon_{\text{mix}} \frac{1}{N}.$$

Each problem therefore retains a minimum sampling probability of ϵ_{mix}/N.

Insight retrieval. Given a sampled problem q, CORE retrieves insights in two steps. First, CORE retrieves a broad set of previously encountered training problems that are semantically similar to q. Let $\mathcal{P}(\mathcal{E})$ denote the set of unique training problems indexed in rollout memory. We embed q as e_q and define $\mathcal{N}_q$ as the set of Z nearest training problems under cosine similarity:

$$\mathcal{N}_q = \{p_1, \ldots, p_Z\}, \qquad s(e_q, e_{p_1}) \geq \cdots \geq s(e_q, e_{p_Z}),$$

where $s(e_q, e_p)$ denotes cosine similarity between problem embeddings. Then, CORE scores each insight by aggregating its observed utilities over this neighborhood:

$$\hat{U}(q, i) = \frac{\sum_{p \in \mathcal{N}_q : (p,i) \text{ observed}} N_{p,i} \bar{U}_{p,i}}{\sum_{p \in \mathcal{N}_q : (p,i) \text{ observed}} N_{p,i} + \lambda}.$$

Weighting by $N_{p,i}$ gives more influence to problem–insight pairs with more observations. If insight i has not been applied to any problem in $\mathcal{N}_q$, its local utility estimate is set to zero before exploration bonuses are applied. Exploration bonuses are defined as $B(i) = \beta \sqrt{\frac{\log(T+1)}{N_{\text{global}}(i)+1}}$, where $N_{\text{global}}(i)$ is the total number of retrievals for insight i during training, T is the total number of retrievals for all insights during training, and β controls the strength of exploration. The retrieval score during training is thus $S_{\text{train}}(q, i) = \hat{U}(q, i) + B(i)$. CORE uses this score to retrieve the top-K insights and appends them to the solver prompt for M. M then attempts to solve q, and the verifier returns a reward r that CORE uses to update the utility score of each retrieved insight $i \in I$ using the baseline-relative utility $U(q)$. Specifically,

$$N_{q,i} \leftarrow N_{q,i} + 1, \qquad \bar{U}_{q,i} \leftarrow \bar{U}_{q,i} + \frac{U(q) - \bar{U}_{q,i}}{N_{q,i}}.$$

CORE also increments $N_{\text{global}}(i)$ and T, and stores the newly generated rollout in rollout memory. This update assigns the same observed utility to every insight retrieved for a solve, yielding a group-level credit assignment rule. CORE addresses this limitation through its admission procedure: new insights are tested individually before entering insight memory, while later retrieval updates estimate where admitted insights are useful across many problems.

Contrastive reflection. When CORE fails to correctly solve a problem during training, it triggers a contrastive reflection step. Here, the failed attempt becomes the negative rollout τ^-. To obtain a positive rollout τ^+, CORE retrieves the most similar correct rollout from rollout memory, as determined by problem embeddings. Importantly, τ^+ can be for the same problem as τ^- if that

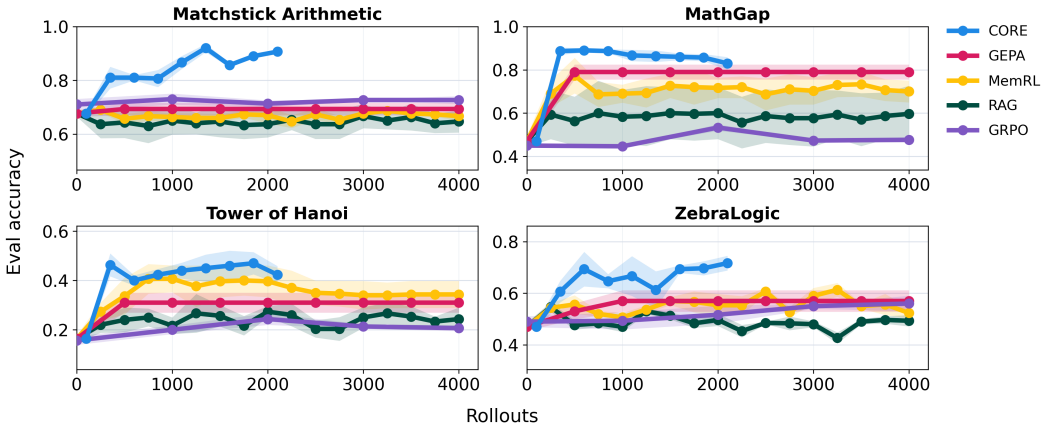


Figure 2: **CORE improves rollout efficiency.** Held-out evaluation performance as a function of total training rollouts in the 10-example regime. Across all four tasks, CORE learns more quickly and more effectively than GEPA, MemRL, RAG, and GRPO.

problem has been correctly solved before. If rollout memory has multiple rollouts for the selected problem, then CORE will choose the one with the most similar reasoning trace embedding. If rollout memory does not yet have any correct rollouts, then CORE will skip contrastive reflection altogether.

Given (τ^-, τ^+), the same frozen model M is prompted to generate a small set of candidate insights. Because a rollout may be successful or unsuccessful due to subtle reasons, CORE therefore makes it easier for the model to surface meaningful insights by retrieving τ^- and τ^+ that are similar.

CORE filters out duplicate and pre-existing candidate insights and then admission-tests the remaining candidates before they are added to the insight memory. During admission-testing, CORE prompts M with each candidate i as the only in-context insight and samples n_{adm} solutions for the originating problem q. With $\hat{a}_{q,i}$ denoting the success rate, CORE admits i into insight memory only if $\hat{a}_{q,i} > b_q + \delta$, where $\delta \geq 0$ is an admission margin. In our experiments, we use $n_{\text{adm}} = 1$ and $\delta = 0$, so a candidate insight is admitted if it leads the model to solve the problem on the first attempt. Admitted insights are initialized with the utility scores that were observed during their admission trial, and rejected insights are discarded. We consider the effect of noisy insights in Appendix D.

Evaluation. During evaluation, both memories are frozen. For each held-out problem $q \in \mathcal{D}_{\text{eval}}$, CORE retrieves the top-K insights using the exploitation-only score $S_{\text{eval}}(q, i) = \hat{U}(q, i)$, with no exploration bonus, reflection, admission testing, or memory updates. Test time retrieval uses only training problems stored in rollout memory while test time rollouts are never added to either memory.

4 Evaluation

4.1 Setup

Tasks. We evaluate CORE on four verifiable reasoning tasks spanning algorithmic, arithmetic, logical, and symbolic problem solving. We selected these tasks because they remain unsaturated on OpenAI `gpt-oss-120b`, the model used in our experiments(Agarwal et al., 2025). **Tower of Hanoi** requires generating valid action sequences for a classic planning problem with recursive structure (Shojaee et al., 2025). **MathGAP** consists of arithmetic word problems with controllable proof structure and complexity (Opedal et al., 2024). **ZebraLogic** includes logic-grid puzzles from constraint-satisfaction problems with controllable search complexity (Lin et al., 2025). **Matchstick arithmetic** consists of matchstick equation puzzles, a classic in the study of human insight problem solving (Knoblich et al., 1999), with specific problems adopted from (Nasvytis & Fan, 2026).

Training. For each of the above tasks, we run evaluations on CORE and our baseline algorithms using training sets of 5, 10, and 100 problems. In all regimes, held-out performance is measured on a

Method	Matchstick Arithmetic			MathGAP		
	5 train items	10 train items	100 train items	5 train items	10 train items	100 train items
No Learning	0.681 ± 0.007	0.681 ± 0.007	0.681 ± 0.007	0.472 ± 0.008	0.472 ± 0.008	0.472 ± 0.008
GRPO	0.630 ± 0.020	0.637 ± 0.022	0.590 ± 0.006	0.393 ± 0.022	0.400 ± 0.023	0.443 ± 0.012
GEPA	0.687 ± 0.019	0.693 ± 0.023	0.770 ± 0.079	0.853 ± 0.027	0.790 ± 0.035	0.777 ± 0.003
Episodic RAG	0.703 ± 0.066	0.627 ± 0.041	0.640 ± 0.038	0.770 ± 0.035	0.590 ± 0.126	0.710 ± 0.084
MemRL	0.700 ± 0.015	0.647 ± 0.012	0.703 ± 0.050	0.747 ± 0.043	0.713 ± 0.062	0.833 ± 0.015
CORE (ours)	$\mathbf{0.873 \pm 0.023}$	$\mathbf{0.907 \pm 0.003}$	$\mathbf{0.870 \pm 0.010}$	$\mathbf{0.873 \pm 0.009}$	$\mathbf{0.830 \pm 0.029}$	$\mathbf{0.843 \pm 0.009}$

Method	Tower of Hanoi			ZebraLogic		
	5 train items	10 train items	100 train items	5 train items	10 train items	100 train items
No Learning	0.179 ± 0.007	0.179 ± 0.007	0.179 ± 0.007	0.509 ± 0.014	0.509 ± 0.014	0.509 ± 0.014
GRPO	0.077 ± 0.009	0.120 ± 0.006	0.107 ± 0.027	0.523 ± 0.027	0.533 ± 0.027	0.520 ± 0.010
GEPA	0.433 ± 0.103	0.310 ± 0.040	0.353 ± 0.185	0.597 ± 0.068	0.570 ± 0.042	$\mathbf{0.707 \pm 0.015}$
Episodic RAG	0.287 ± 0.003	0.243 ± 0.050	0.303 ± 0.048	0.540 ± 0.015	0.493 ± 0.032	0.497 ± 0.015
MemRL	$\mathbf{0.517 \pm 0.069}$	0.393 ± 0.047	$\mathbf{0.727 \pm 0.100}$	0.683 ± 0.012	0.543 ± 0.052	0.587 ± 0.020
CORE (ours)	0.400 ± 0.049	$\mathbf{0.423 \pm 0.035}$	0.427 ± 0.058	$\mathbf{0.700 \pm 0.036}$	$\mathbf{0.717 \pm 0.028}$	0.663 ± 0.022

Table 1: **CORE improves sample efficiency**. Final held-out accuracy across training-example regimes. Entries show mean $\pm$ SEM across completed runs. The no-learning baseline is averaged across rollout-0 evaluations from all training-size regimes. Bold indicates the highest mean accuracy for each task and training-size regime.

separate evaluation set of 100 problems. This design allows us to measure how well each algorithm can support robust learning given small training sets. For each task, method, and training-set size, we run three independent training runs and report mean held-out verifier accuracy.

Model. We use OpenAI `gpt-oss-120b` (Agarwal et al., 2025) in all experiments and for all learning algorithms. Unless otherwise noted, generation uses temperature 0.6, maximum output length $32{,}768$ tokens and reasoning effort set to `high`. For MemRL, following the authors' recommendations, memory-building and adjustment calls use temperatures 0.0 and 0.3, respectively, and memory retrieval uses OpenAI `text-embedding-3-large` embeddings. For CORE and Episodic RAG text embeddings, we use `jina-embeddings-v2-base-en` (Günther et al., 2023). For inference, we use NVIDIA NIM and Cerebras, matching all model-facing parameters across backends and verifying that baseline held-out accuracies were comparable across all tasks. For CORE, we retrieve top-$K_{\text{train}} = K_{\text{eval}} = 25$ insights per task. In Appendix B we also replicate our main experiments using Qwen 3.5 9B (Qwen Team, 2026) and show that CORE generalizes to other and smaller models.

Baselines. We compare CORE against three non-parametric baselines and one parametric baseline, chosen to represent strong alternative approaches to learning from verifiable rewards: **GRPO** (Guo et al., 2025) to represent parametric RLVR, **GEPA** (Agrawal et al., 2025) to represent prompt optimization, **Episodic RAG** to represent retrieving entire past rollouts, and **MemRL** (Zhang et al., 2026b) to represent value-aware episodic memory. Our GEPA implementation uses the standard DSPy implementation and our MemRL implementation is adapted from the official codebase and uses the strongest public configuration. Our RAG implementation is a setting where all rollouts are added to a rollout memory and indexed according to their problem embedding. For any model inference during training and testing, we retrieve the most similar past successful rollout and the most similar past unsuccessful rollout to place in-context alongside their verifier feedback. If the rollout memory does not yet contain any successful or unsuccessful rollouts, then we retrieve similar rollouts without accounting for success. Finally, our GRPO implementation uses LoRA via the Tinker API (Lab, 2025) with a rank of 32, a learning rate of $1e-5$, a batch size of 10, and a group size of 10. In the following sections we describe CORE's advantages over these baselines with respect to rollout, sample, and context efficiency, but we also provide cost and token count breakdowns in Appendix C.

4.2 CORE ACHIEVES STRONGER PERFORMANCE WITH FEWER ROLLOUTS

We first evaluate CORE's *rollout efficiency*, the number of rollouts needed to achieve meaningful task improvement, in the 10-sample training regime. To ensure a fair comparison, we count CORE's

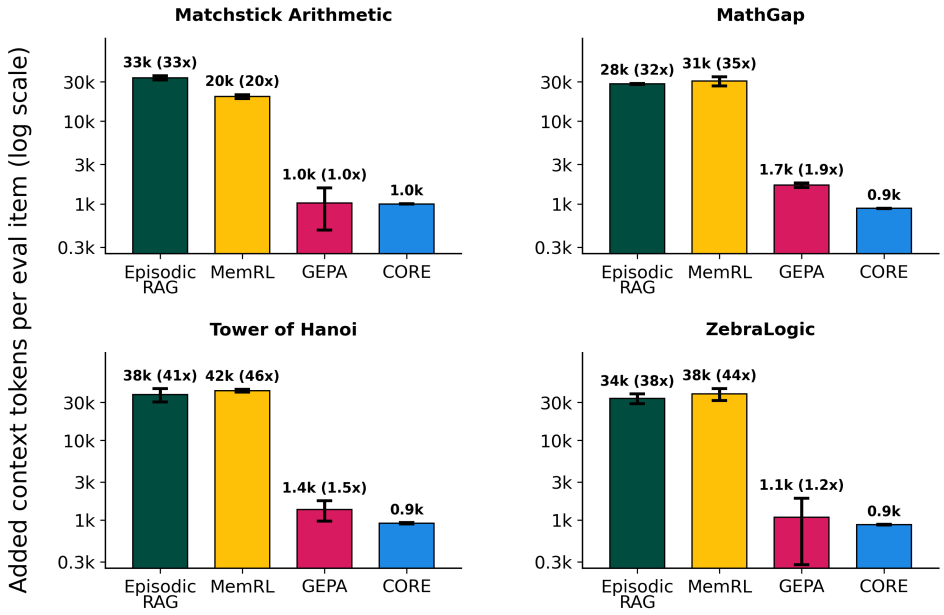


Figure 3: **CORE improves context efficiency.** Added task-external context tokens per evaluation item for each method, averaged across the 5-, 10-, and 100-sample training regimes (lower number denotes higher efficiency). Error bars show SEM across training-data regimes. Labels above bars report mean added tokens and the fold increase relative to CORE. **The y-axis is log-scaled**.

initial baseline accuracy runs and insight admission tests as rollouts. Results are displayed in Figure 2. CORE learns rapidly across all tasks: by 350 training rollouts it exceeds the best performance achieved by any baseline at any point. CORE also settles on higher final performance than any baseline despite the fact that we run it for 2100 training rollouts compared to 4000 for our baselines.

Averaged across tasks, CORE's held-out accuracy improves by 59.9% from rollout 0 to 350, increasing from 0.445 to 0.712, and sustains its gains until rollout 2,100[1], achieving a 0.717 held-out accuracy. For individual tasks, CORE improves from rollout 0 to 2100 by 34.5% on Matchstick Arithmetic, 76.6% on MathGAP, 159.2% on Tower of Hanoi, and 50.0% on ZebraLogic. These results suggest that CORE can learn from a small number of training samples quickly and effectively.

4.3 CORE ACHIEVES STRONGER PERFORMANCE WITH FEWER TRAINING SAMPLES

We next evaluate CORE's *sample efficiency* across different training data regimes. Specifically, we compare CORE against our baselines under fixed rollout budgets while varying the number of training samples. Table 1 illustrates held-out evaluation performance when each method is trained with 5 training samples for 2050 rollouts, 10 samples for 2100 rollouts, and 100 samples for 3000 rollouts[2].

CORE achieves the highest mean held-out accuracy in 9 of 12 task-by-data-regime conditions. Averaged across tasks, CORE improves over the no-learning baseline by 54.8%, 56.2%, and 52.3% with 5, 10, and 100 training examples, respectively. The only conditions in which CORE does not achieve the highest mean held-out accuracy across all other baselines are Tower of Hanoi with 5 and 100 training samples (where MemRL achieves the highest accuracy), and ZebraLogic with 100

[1]For CORE, rollout counts include the initial 100 rollouts used to estimate per-problem baseline accuracies, so the first evaluation occurs at 350 training rollouts and the final evaluation occurs at 2100 rollouts.

[2]In settings with different numbers of training samples, the initial number of rollouts that CORE uses to calculate per-problem baseline accuracies is calculated as number of training samples multiplied by ten. To give our baseline methods a better chance, we allow them to train for these additional rollouts as well, which is why we report results for 2050, 2100, and 3000 rollouts. For all data regimes, we only run CORE for 2000 actual training rollouts.

Insight type	What it captures	Example learned insight
Search space structuring	Identifies broad constraints, search heuristics, or specific reusable solution patterns that reduce the number of possibilities to consider.	*Matchstick Arithmetic:* "Use chain equality: converting a minus into a second = can create a three-part equality, allowing the equation to be solved when the three terms become identical." (Utility: 0.14)
Intermediate state tracking	Keeps intermediate steps, quantities, or moves updated as each step changes the problem.	*MathGAP:* "When a clue describes a transfer or split, update the giver's and all receivers' counts immediately, and keep pre- and post-transfer values separate for later use." (Utility: 0.09)
Verification and validation	Checks candidate solutions against task constraints, detects contradictions, backtracks, or satisfies output requirements.	*Tower of Hanoi:* "Before finalizing your answer, simulate each move step-by-step, confirming that the move obeys the one-disk, top-disk, and size-order rules, and record the resulting peg states." (Utility: 0.11)

Table 2: Examples of high-utility insights learned by CORE, grouped by functional role. Utilities denote the empirical baseline-relative utility associated with each insight.

training samples (where GEPA achieves the highest accuracy). Overall, these results suggest that CORE can extract insights from training sets of different sizes across diverse reasoning tasks.

4.4 CORE ADDS SIGNIFICANTLY LESS EVALUATION-TIME CONTEXT

We also evaluate *context efficiency*: how many additional tokens each non-parametric method and CORE adds to the evaluation prompt beyond the base task prompt and answer-format instructions. Figure 3 demonstrates that CORE is the most context-efficient method we evaluate. Averaged across tasks, CORE adds 0.92k tokens per evaluation item, compared with 33.6k for Episodic RAG and 32.7k for MemRL. CORE is about $1.4\times$ more context-efficient than GEPA, a meta-prompt optimization baseline, which adds an average of 1.29k tokens per evaluation item.

Thus, CORE's gains do not come from placing large retrieved traces into the evaluation prompt; instead, it compresses training-time experience into a small set of abstract, reusable insights. Taken together, these findings suggest that CORE not only outperforms baseline algorithms across the vast majority of task and data regimes, but does so while adding substantially less evaluation-time context.

4.5 CORE PRIORITIZES INSIGHTS THAT CAPTURE RECURRING REASONING PATTERNS

Finally, we highlight that CORE stores learning artifacts as natural-language insights. To characterize these artifacts, we examine one run from each task in the 10-training-example regime. We analyze how CORE's memory grows, how insight utility is distributed, and what types of insights it contains.

Memory growth. Averaged across all CORE runs, after 2,000 rollouts the number of admitted insights was 143 for Matchsticks, 126 for ZebraLogic, 119 for Tower of Hanoi, and 65 for MathGAP.

Utility distribution. Insight utility is concentrated unevenly across memory (Figure 6). Most admitted insights have non-negative estimated utility, with non-negative weighted mass above 91% for all tasks. The distributions are approximately unimodal for Matchstick Arithmetic and ZebraLogic, but bimodal for MathGAP and Tower of Hanoi. This pattern suggests that CORE admits many mildly useful insights, while a smaller subset of high-utility insights drives the largest gains.

Analysis of insights. To understand what CORE learns, we manually inspected high-utility insights from each task and grouped near-duplicates by functional role (Table 2). Insights fell into three broad categories: *search-space structuring*, which includes both broad heuristics and specific reusable solution patterns; *intermediate-state tracking*, which keeps quantities, assignments, moves, or equations updated as reasoning unfolds; and *verification and validation*, which checks constraints, detects contradictions, and enforces output requirements. These patterns suggest that CORE stores procedural abstractions for guiding future reasoning, rather than summaries of prior episodes.

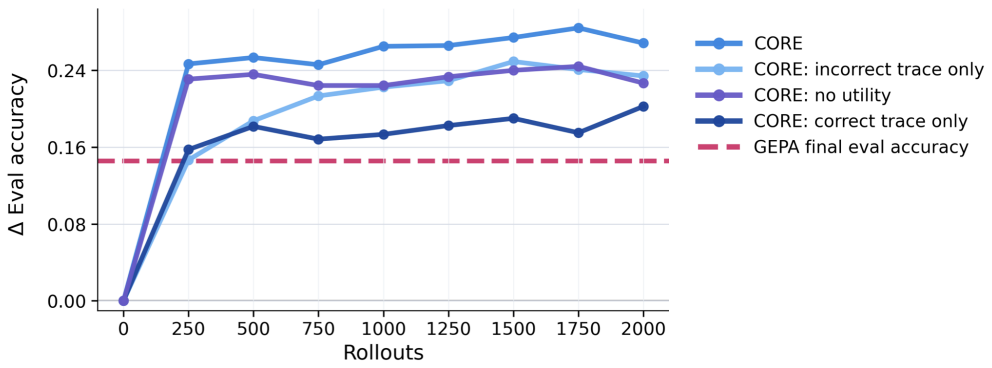


Figure 4: Ablations in the 10-training-example setting, aggregated across all four tasks. Y-axis shows improvement in held-out accuracy relative to rollout 0. *Incorrect trace only* reflects only on the model's most recent incorrect reasoning trace, rather than comparing failure against success. *Correct trace only* generates insights from a single successful trace, using a correct trace from the same problem when available. *No utility* keeps contrastive insight generation but retrieves insights using only semantic similarity, without utility estimates. GEPA denotes the strongest non-CORE baseline.

5 What Accounts for Learning from Contrastive Reflection?

We use ablations in the 10-example setting across all four tasks to test which components of CORE drive its learning gains. The ablations ask two questions: whether contrastive reflection improves the generation of useful insights, and whether empirical utility estimates improve their retrieval.

Does contrastive reflection improve insight generation? CORE generates insights by comparing a failed reasoning trace with a successful one from the same or a similar problem. We compare this with two non-contrastive variants: one that reflects only on the most recent incorrect trace (*incorrect trace only*), and one that reflects only on a correct trace (*correct trace only*). Aggregating across all four tasks, CORE achieves the highest improvement over rollout 0 at every nonzero evaluation checkpoint, reaching a final mean improvement of 0.268, compared with 0.234 for incorrect-only reflection and 0.203 for correct-only reflection. This suggests that the strongest insights come from explicitly comparing failure and success, rather than reflecting on either trajectory in isolation.

Does utility-aware retrieval improve insight reuse? CORE retrieves insights using both semantic relevance and learned utility estimates. We compare this with a variant that retrieves insights using only the semantic similarity between the problems to which an insight has been applied and the current problem. Removing utility-aware retrieval reduces the final mean improvement from 0.268 to 0.227, indicating that relevance alone is not sufficient for best performance.

All CORE variants outperform GEPA, the strongest baseline, in final held-out accuracy, but full CORE achieves the largest gains. This suggests that reusable insight generation is itself a strong learning mechanism, and that it performs best with both contrastive reflection and utility-aware reuse.

6 Conclusion

We present CORE, a non-parametric learning algorithm that improves sample, rollout, and context efficiency over strong baselines across four logic, planning, and problem-solving tasks. The main implication of CORE is that learning from experience can be made more efficient by changing *what* is stored and reused. While standard episodic-memory methods store and retrieve entire rollouts or summaries of rollouts, CORE instead stores *insights* about rollouts: it contrasts successful and failed rollouts to produce candidate insights gated through verifier feedback and retrieved using semantic relevance and observed utility. In this sense, CORE uses reflection as a means to perform credit assignment, with insights being hypotheses about which strategies to carry forward rather than a summary of what happened. Because these artifacts are inspectable and paired with empirical utility estimates, they are more transparent than weight updates and prompt optimization, with potential relevance to concerns about opaque self-improvement (Korbak et al., 2025; Emmons et al., 2025).

AI USE STATEMENT

We used OpenAI Codex and Claude Code for coding assistance. Code was then verified manually. We also used ChatGPT and Claude for occasional help with conducting literature review. Aside from these use cases, all ideas, writing, and figures were human generated.

REFERENCES

R Alison Adcock, Arul Thangavel, Susan Whitfield-Gabrieli, Brian Knutson, and John DE Gabrieli. Reward-motivated learning: mesolimbic activation precedes memory formation. *Neuron*, 50(3): 507–517, 2006.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

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. *arXiv preprint arXiv:2507.19457*, 2025.

Louis Alfieri, Timothy J Nokes-Malach, and Christian D Schunn. Learning through case comparisons: A meta-analytic review. *Educational Psychologist*, 48(2):87–113, 2013.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Fiery Cushman. Rationalization is rational. *Behavioral and Brain Sciences*, 43:e28, 2020.

Aniket Didolkar, Nicolas Ballas, Sanjeev Arora, and Anirudh Goyal. Metacognitive reuse: Turning recurring llm reasoning into concise behaviors. *arXiv preprint arXiv:2509.13237*, 2025.

Scott Emmons, Erik Jenner, David K Elson, Rif A Saurous, Senthooran Rajamanoharan, Heng Chen, Irhum Shafkat, and Rohin Shah. When chain of thought is necessary, language models struggle to evade monitors. *arXiv preprint arXiv:2507.05246*, 2025.

Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, et al. Jina embeddings 2: 8192-token general-purpose text embeddings for long documents. *arXiv preprint arXiv:2310.19923*, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Günther Knoblich, Stellan Ohlsson, Hilde Haider, and Detlef Rhenius. Constraint relaxation and chunk decomposition in insight problem solving. *Journal of Experimental Psychology: Learning, memory, and cognition*, 25(6):1534, 1999.

Tomek Korbak, Mikita Balesni, Elizabeth Barnes, Yoshua Bengio, Joe Benton, Joseph Bloom, Mark Chen, Alan Cooney, Allan Dafoe, Anca Dragan, et al. Chain of thought monitorability: A new and fragile opportunity for ai safety. *arXiv preprint arXiv:2507.11473*, 2025.

Thinking Machines Lab. Tinker, 2025. URL `https://thinkingmachines.ai/tinker/`.

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 *The twelfth international conference on learning representations*, 2023.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*, 2025.

John E Lisman and Anthony A Grace. The hippocampal-vta loop: controlling the entry of information into long-term memory. *Neuron*, 46(5):703–713, 2005.

Andrew J Nam and James L McClelland. Systematic human learning and generalization from a brief tutorial with explanatory feedback. *Open Mind*, 8:148–176, 2024.

Linas Nasvytis and Judith E Fan. Leveraging speech to identify signatures of insight and transfer in problem solving. *arXiv preprint arXiv:2605.12970*, 2026.

Andreas Opedal, Haruki Shirakami, Bernhard Schölkopf, Abulhair Saparov, and Mrinmaya Sachan. Mathgap: Out-of-distribution evaluation on problems with arbitrarily complex proofs. *arXiv preprint arXiv:2410.13502*, 2024.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, 2024.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

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

Daniel L Schwartz and John D Bransford. A time for telling. *Cognition and instruction*, 16(4): 475–522, 1998.

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.

Taiwei Shi, Sihao Chen, Bowen Jiang, Linxin Song, Longqi Yang, and Jieyu Zhao. Experiential reinforcement learning. *arXiv preprint arXiv:2602.13949*, 2026.

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.

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*, 2025.

Yuda Song, Lili Chen, Fahim Tajwar, Remi Munos, Deepak Pathak, J Andrew Bagnell, Aarti Singh, and Andrea Zanette. Expanding the capabilities of reinforcement learning via text feedback. *arXiv preprint arXiv:2602.02482*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *arXiv preprint arXiv:2602.12430*, 2026.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Dylan Zhang, Yanshan Lin, Zhengkun Wu, Yihang Sun, Bingxuan Li, Dianqi Li, and Hao Peng. Useful memories become faulty when continuously updated by llms. *arXiv preprint arXiv:2605.12978*, 2026a.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026b.

APPENDIX OUTLINE

- Pseudocode for the CORE algorithm.
- Results for Qwen 3.5 9B
- Additional compute accounting
- Resilience to a stricter admission parameter
- Growth of the insight memory across training rollouts.
- Distributions of insight utilities.

A PSEUDOCODE FOR THE CORE ALGORITHM

Algorithm 1 gives pseudocode for the CORE training procedure, including failure-biased sampling, insight retrieval, contrastive reflection, and admission testing.

Algorithm 1 CORE Training

Require: Training set $\mathcal{D}_{\text{train}}$, frozen model M, verifiers $\{V_q\}$, rollout memory $\mathcal{R}$, insight memory $\mathcal{I}$

Initialize $\mathcal{R} \leftarrow \emptyset, \mathcal{I} \leftarrow \emptyset$
Estimate initial no-memory baselines b_q for each $q \in \mathcal{D}_{\text{train}}$
for $t = 1, \ldots, T_{\text{train}}$ **do**
    Sample training problem q using failure-biased sampling
    Retrieve top-K insights L from $\mathcal{I}$ using neighbors of q in $\mathcal{R}$
    Generate solution $y \leftarrow M(q, L)$ and reward $r \leftarrow V_q(y)$
    Compute utility $U(q) \leftarrow r - b_q$
    Update insight-memory statistics for each $\ell \in L$ using $U(q)$
    Store rollout $\tau = (q, L, y, r)$ in rollout memory $\mathcal{R}$
    if $r = 0$ **then**
        Retrieve positive rollout τ^+ from $\mathcal{R}$
        if τ^+ exists **then**
            Generate candidate insights $C \leftarrow M(\tau, \tau^+)$
            Filter candidate insights C
            for all $\ell \in C$ **do**
                Admission-test ℓ on q
                if ℓ passes admission **then**
                    Add ℓ to insight memory $\mathcal{I}$
                    Initialize utility statistics for ℓ
                end if
            end for
        end if
    end if
end for

B RESULTS FOR QWEN 3.5 9B

Here we demonstrate that CORE transfers to a smaller model and remains competitive with matched baselines. To do so we reproduce our experiments on Qwen 3.5 9B (Qwen Team, 2026). We use the same frozen 9B model for solving, reflection, and insight generation. For sampling, we use a temperature of 1.0, top p of 0.95, top k of 20 and no presence penalty for all experiments. For GRPO, we finetune the model with a learning rate of $1e-6$, weight decay of 0.1, batch size of 100, and KL loss. During evaluation, we set a token limit of 32k for all experiments.

Table 3 shows how CORE compares to our baselines when each is applied to Qwen 3.5 9B. For sample efficiency, we can observe that CORE achieves higher mean accuracy than GEPA in eight out of twelve conditions and beats Episodic RAG, MemRL and budget-matched GRPO in all conditions. Core also has the highest mean accuracy in all four of the 5-example conditions. For rollout efficiency, we can compare CORE with our baselines in the 10-example training regime. At 2,100 training rollouts, CORE achieves higher mean accuracy than Episodic RAG, MemRL, and GRPO on all four tasks, and a higher mean than GEPA on two of the four tasks. We therefore conclude that CORE and its main benefits to transfer to other models and also smaller models.

Table 3: Performance comparison between CORE and our baselines across tasks and numbers of training examples. Results are reported as mean ± standard deviation. The best-performing method is bolded in each row.

Task · examples	No learning	CORE	GEPA	Episodic RAG	MemRL	GRPO (matched)
Matchstick · 5	42.0 ± 1.0	$\mathbf{81.7 \pm 4.4}$	70.7 ± 1.2	56.3 ± 2.0	52.0 ± 3.0	50.0 ± 1.5
Matchstick · 10	42.0 ± 1.0	$\mathbf{86.7 \pm 2.2}$	76.0 ± 4.6	53.3 ± 6.2	67.3 ± 3.3	45.7 ± 4.9
Matchstick · 100	42.0 ± 1.0	84.0 ± 4.6	$\mathbf{95.0 \pm 0.6}$	58.7 ± 0.3	62.3 ± 0.3	50.7 ± 0.9
MathGAP · 5	27.3 ± 1.3	$\mathbf{78.7 \pm 0.9}$	64.7 ± 16.8	50.0 ± 4.0	52.0 ± 0.0	60.7 ± 3.2
MathGAP · 10	27.3 ± 1.3	76.3 ± 3.4	$\mathbf{80.3 \pm 4.3}$	35.3 ± 3.5	43.7 ± 2.2	64.3 ± 3.8
MathGAP · 100	27.3 ± 1.3	79.3 ± 2.7	$\mathbf{86.3 \pm 1.8}$	39.0 ± 3.6	59.3 ± 5.0	64.0 ± 6.4
ZebraLogic · 5	72.3 ± 1.2	$\mathbf{76.3 \pm 2.0}$	75.3 ± 2.3	54.7 ± 1.9	56.7 ± 4.6	34.3 ± 8.8
ZebraLogic · 10	72.3 ± 1.2	$\mathbf{78.7 \pm 0.3}$	73.0 ± 1.5	54.0 ± 9.5	57.7 ± 1.5	32.0 ± 19.5
ZebraLogic · 100	72.3 ± 1.2	$\mathbf{76.7 \pm 2.8}$	76.3 ± 1.2	59.0 ± 1.5	54.0 ± 1.7	18.0 ± 7.9
Tower of Hanoi · 5	16.3 ± 0.7	$\mathbf{19.3 \pm 0.3}$	15.8 ± 0.6	16.0 ± 1.5	11.3 ± 3.0	15.3 ± 2.3
Tower of Hanoi · 10	16.3 ± 0.7	19.3 ± 0.9	$\mathbf{20.0 \pm 1.5}$	14.7 ± 0.9	11.7 ± 1.2	9.3 ± 0.9
Tower of Hanoi · 100	16.3 ± 0.7	$\mathbf{18.3 \pm 1.2}$	18.0 ± 1.2	15.0 ± 1.0	11.3 ± 0.3	11.5 ± 0.5

C Additional compute accounting

On GPT-OSS-120B, after counting input and output tokens from every training-time LLM call, CORE first exceeds the best accuracy achieved by any baseline at 10.0M cumulative training tokens on Matchstick, 8.0M on MathGAP, 9.8M on Tower of Hanoi, and 26.3M on ZebraLogic. These target accuracies are set by GRPO on Matchstick, GEPA on MathGAP, and MemRL on Tower of Hanoi and ZebraLogic; exceeding them also exceeds RAG and every other baseline. Our original rollout counts included baseline estimation and admission tests. However, following the convention set by GEPA (Agrawal et al., 2025), we excluded reflection/proposal LM calls and apply the same convention to our accounting of GEPA proposals and MemRL summaries. We also find that when we count the exact input and output tokens for every LLM call, reflection accounts for 31% of CORE's training tokens while embedding and retrieval use no LLM tokens and together account for less than 1% of training compute.

In terms of dollar cost, mean reference cost per run at the matched 2,100-rollout point in the 10 training example regime is 29 USD for CORE, 45 USD for GRPO, 17 USD for RAG, 34 USD for MemRL, and 0 USD cached or 22 USD cacheless for GEPA. CORE is thus the most accurate method on every task at this point and never the most expensive.

GEPA's cached and cacheless figures describe the same runs: the cached view counts only calls that generate new tokens, while the cacheless view also charges replayed DSPy evaluations. CORE exceeds GEPA's final accuracy within 8–15M cumulative training tokens under either view. We do not compare wall-clock time because the methods used different serving providers. At evaluation, CORE adds the least amount of context tokens on every task: 0.92k tokens per item compared with 33.6k for Episodic RAG, 32.7k for MemRL, and 1.29k for GEPA.

D Resilience to a stricter admission parameter

One potential issue with CORE is that it can admit some insights by chance. To test the effect of varying the admission parameter, we ran the Matchstick Arithmetic experiments with three attempts per candidate ($n_{adm} = 3, \delta = 0$) and three runs per model. The stricter admission gate admits 36% fewer insights on Qwen 3.5 9B and 33% fewer insights on GPT OSS 120B. Yet, mean final accuracy changes only marginally from 86.7 to 84.7 on Qwen 3.5 9B, and from 88.7 to 88.3 on GPT OSS 120B. A stricter admission gate can therefore remove roughly one third of admissions while changing mean Matchstick accuracy by at most 2 points.

E Growth of the insight memory across training rollouts

Figure 5 shows how the number of admitted insights grows across training rollouts for each task. Ribbons show 95% confidence intervals across the runs.

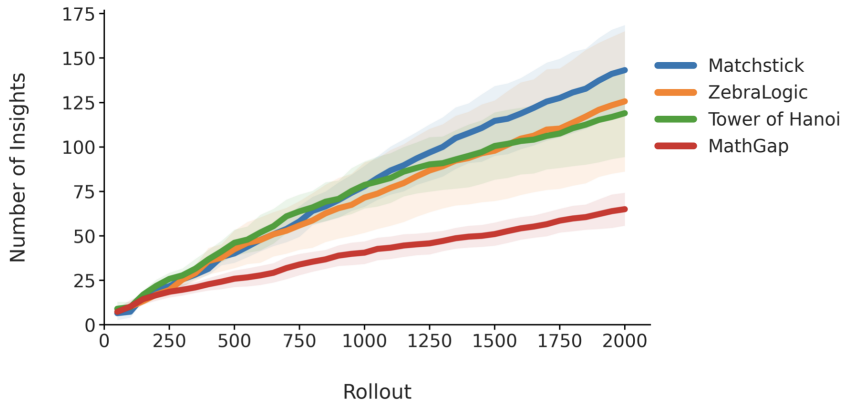


Figure 5: Growth of the insight memory across rollouts for each task. Lines show the number of stored insights accumulated during training. Ribbons show 95% confidence intervals across the runs.

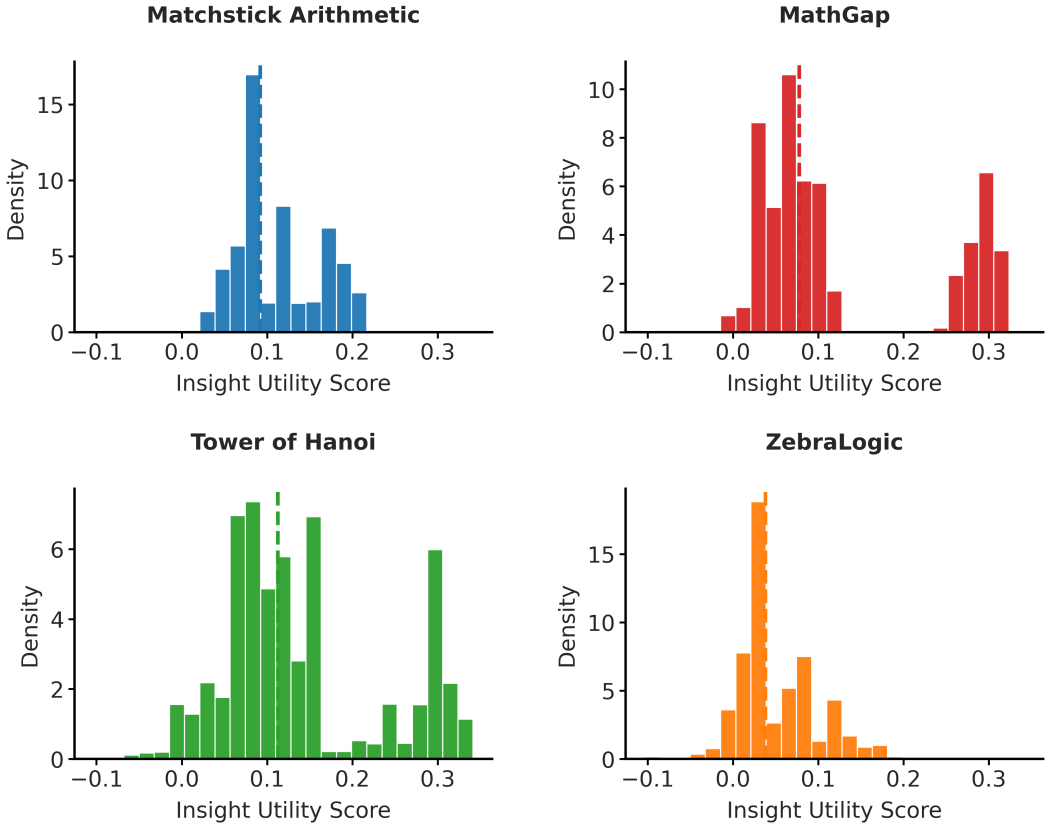


Figure 6: Distributions of insight utilities across the four tasks.

F Distributions of insight utilities

Figure 6 shows the distribution of estimated insight utilities across the four tasks.