LEARN TO REMEMBER: GEOMETRIC MEMORY FOR INFERENCE-TIME SELF-IMPROVEMENT IN LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are used in a stateless way, discarding what they learn from one query to the next. Memory-augmented LLMs adapt at inference time, but existing systems either inject the entire memory into every prompt, so cost grows with it, or retrieve by similarity under a fixed encoder that never learns whether an entry helped. We propose **Le**arn to **Re**member (**LeRe**), in which a frozen LLM cycles through plan, solve, and curate roles and learns which memory to retrieve rather than how much to store. A Planner rewrites each query into a structured retrieval key; the Contrastive Contextual Memory Encoder (CCME), lightweight heads on a frozen encoder, is trained online from the Curator's attribution of which entries helped, so retrieval reflects measured utility; and a Guard–Consolidate–Maintain (GCM) gate rejects answer leakage, merges near-duplicates, and quarantines unreliable entries. LeRe learns without ground-truth labels or weight updates. Across twelve mathematical, scientific, and multimodal streams and three backbones, LeRe improves average accuracy over a no-memory baseline by 6.9%, 9.0%, and 4.6%, exceeding the strongest memory-based baseline on every backbone and gaining up to 24.7% on competition mathematics, while its per-query token cost stays flat as memory grows, at near-lowest cost. These consistent gains suggest that selecting proven memory, rather than accumulating more, offers a scalable, efficient, and effective way to support inference-time adaptation in LLMs. Our code is available in an anonymous repository at `https://anonymous.4open.science/r/learn-to-remember-AD37`.

1 INTRODUCTION

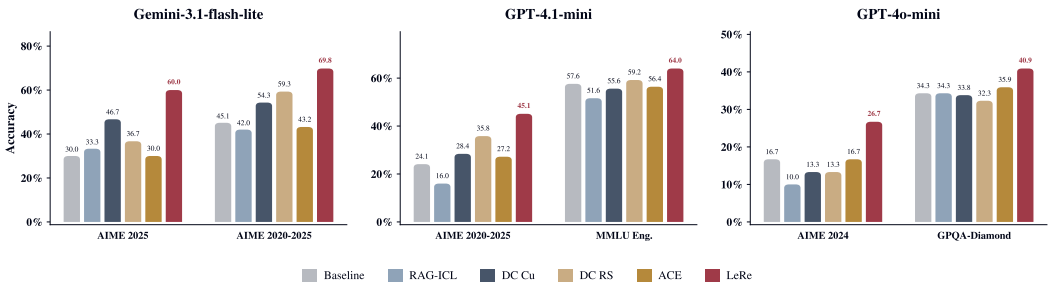


Figure 1: **LeRe against retrieval- and memory-augmented baselines.** Accuracy on two reasoning streams for each of three backbone models. LeRe achieves the highest accuracy on all six, exceeding the strongest competing method by 4.8% to 13.3%.

Large language models (LLMs) have achieved strong empirical success across diverse reasoning benchmarks, including formal mathematics, scientific problem solving, and code synthesis (Brown et al., 2020; Achiam et al., 2023). Nevertheless, standard deployment remains fundamentally stateless: models process successive queries independently, discarding intermediate reasoning, verifica-

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.

tion outcomes, and reusable solution strategies. As a result, they repeat computation, recur in the same failure patterns, and fail to improve over extended interaction horizons.

A growing line of work addresses this through inference-time adaptation with external memory. Recent approaches augment LLMs with non-parametric memory that stores reusable artifacts such as reasoning traces, tool-use patterns, or code snippets (Lewis et al., 2020; Park et al., 2023; Xu et al., 2025). Dynamic Cheatsheet (DC) (Suzgun et al., 2026) and Agentic Context Engineering (ACE) (Zhang et al., 2025) further show that maintaining and reusing such artifacts can improve downstream performance without updating model weights.

As this memory grows without constraint, however, two problems emerge: *redundancy, noise, and drift accumulate*, and *a larger memory often fails to yield better answers*. ACE, for example, places its entire accumulated playbook in the context at every step, so no entry is selected for the query at hand and prompt length grows with the memory. The retrieval variant of DC does select entries per query, but ranks them by similarity in a frozen embedding space (Karpukhin et al., 2020) that never learns whether a retrieved entry actually helped. In both cases, what enters the context is a weak proxy for what improves the answer, which raises two questions:

> **Q1:** Can retrieval be trained to rank memory by its measured contribution to task success, rather than by surface similarity?
>
> **Q2:** How should inference-time memory be organized so that it remains selective and its cost bounded as it grows?

We present **LeRe**, a framework in which a frozen LLM improves itself at inference time by cycling through three agentic roles, ❶ **plan**, ❷ **solve**, and ❸ **curate**, without ground-truth labels or human feedback. Figure 2 illustrates the workflow. A Planner first rewrites each query into a structured retrieval key: a normalized statement of the problem, its domain, and the skills it requires. A **Contrastive Contextual Memory Encoder (CCME)** then scores memory entries by a learned similarity to this key, combined with each entry's reliability, and returns the top-K entries. A Solver conditions on these entries, treated as advisory rather than binding, and commits a reasoning trajectory together with a final answer. A Curator, with no access to ground-truth labels, verifies the answer from execution outcomes or its own judgment, labels each retrieved entry as helpful, harmful, irrelevant, or redundant, and proposes new entries. A **Guard–Consolidate–Maintain (GCM)** gate then rejects any proposal that restates the current problem or its answer, merges near-duplicates into existing entries, and enforces a fixed capacity, so that memory remains selective as it grows. Finally, the Curator's attributions serve as contrastive training pairs that update CCME online over the stream. This design answers both questions. For *Q1*, retrieval improves on two fronts: the Planner's key aligns queries with entries at the level of skills rather than surface wording, and CCME learns which entries actually help. For *Q2*, the Solver never sees the whole memory, only its top-K entries, and GCM bounds how the memory grows, so the cost per query stays flat over the stream.

We evaluate LeRe with three backbone models, Gemini-3.1-flash-lite, GPT-4.1-mini, and GPT-4o-mini, on twelve reasoning streams spanning mathematics (AIME, MATH), scientific knowledge (GPQA-Diamond, MMLU-Pro, HLE), and multimodal reasoning (MathVista, MMMU-Pro), against a no-memory baseline and various memory-based baselines. Our key findings are:

- We propose **LeRe**, a plan–solve–curate framework that pairs a Planner with CCME for retrieval and a Curator with GCM for memory management, adapting a frozen LLM at inference time through a bank of discrete, reliability-tracked entries with bounded capacity.
- We introduce **CCME**, a contrastive encoder trained online from the Curator's attributions without ground-truth labels, so that retrieval reflects the measured utility of each memory entry rather than fixed embedding similarity.
- LeRe consistently outperforms strong baselines, improving average accuracy over the no-memory baseline by 6.9%, 9.0%, and 4.6% on the three backbones and exceeding the strongest baseline by 3.2%, 6.6%, and 1.4%. Gains are largest on reasoning-intensive mathematics, reaching 14.2% to 24.7% on AIME 2020–2025, while its per-query token cost stays flat as memory grows.

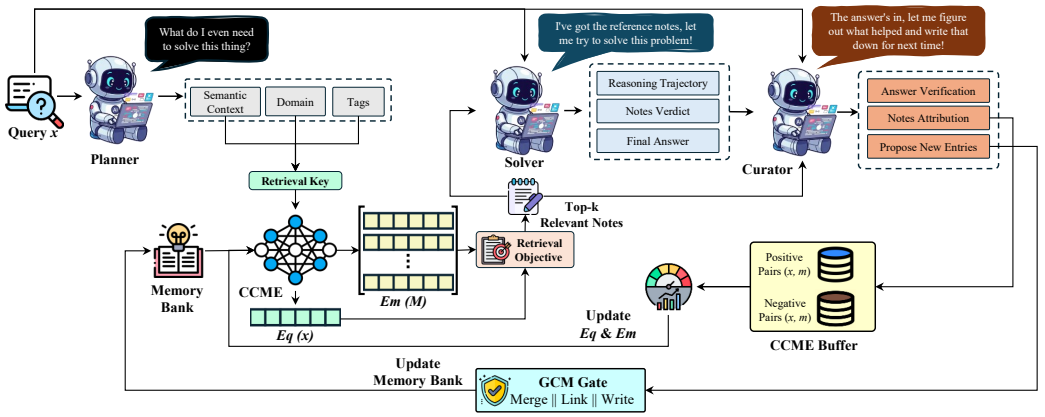


Figure 2: **Overview of the LeRe framework.** A frozen LLM cycles through three agentic roles around a memory bank $\mathcal{M}$ of discrete entries, without ground-truth labels. **(i) Plan:** given a query x, the *Planner* writes a structured retrieval key (semantic context, domain, and tags); CCME encodes it as $E_q(x)$ and the entries as $E_m(\mathcal{M})$, and the retrieval objective combines their learned similarity with each entry's reliability to return the top-K entries. **(ii) Solve:** the *Solver* conditions on x and the retrieved entries, which are advisory rather than binding, and outputs a reasoning trajectory, a verdict on each entry, and a final answer. **(iii) Curate:** the *Curator* verifies the answer from execution outcomes or its own judgment, attributes utility to each retrieved entry, and proposes new entries. Attributed entries form positive and negative pairs (x, m) in the CCME buffer, which update E_q and E_m online, while the GCM gate rejects proposals that restate the problem or its answer and merges, links, or writes the rest into the memory bank.

2 Related Work

Memory-Augmented LLM Agents. Test-time adaptation updates model weights on the incoming stream (Wang et al., 2021; Niu et al., 2022), even without labels (Zuo et al., 2025), which is costly for large models and unavailable for API-served ones; memory-based methods instead condition a frozen model on external memory. Beyond retrieval over a static corpus (Lewis et al., 2020), agentic systems accumulate memory from their own experience, as reflections (Shinn et al., 2023), cross-task insights (Zhao et al., 2024), workflows (Wang et al., 2024c), thought templates (Yang et al., 2024), linked notes (Xu et al., 2025), or paged context (Packer et al., 2023); ReasoningBank (Ouyang et al., 2026) distills strategies from self-judged outcomes without ground-truth labels, and Dynamic Cheatsheet (Suzgun et al., 2026) and ACE (Zhang et al., 2025) curate strategies and code over streams of reasoning problems. These systems invest in what to write but read memory either in full (ACE, DC Cu) or by fixed-encoder similarity (DC RS and most others); ACE's per-bullet helpful and harmful counts serve curation, not retrieval. LeRe differs on both sides: on the write side, its GCM gate rejects answer leakage, merges near-duplicates, and quarantines unreliable entries, keeping memory selective as it grows; on the read side, learned top-K retrieval keeps per-query cost flat as retrieval improves.

Learned Retrieval. Planning before acting (Wang et al., 2023; Yao et al., 2022) and reformulating queries, as hypothetical answers (Gao et al., 2023) or search-oriented rewrites (Ma et al., 2023), improve what is retrieved, and the matching itself can be learned. Dense retrievers are trained contrastively (Oord et al., 2018) on labeled query–passage pairs (Karpukhin et al., 2020); EPR (Rubin et al., 2022), LLM-R (Wang et al., 2024a), and REPLUG (Shi et al., 2024) train retrievers from the likelihood a frozen LM assigns to the gold output. These need gold outputs and token likelihoods, which chat APIs do not return for a supplied answer, and train offline over a fixed pool; Self-RAG (Asai et al., 2024) instead fine-tunes the LM to decide when to retrieve. Closest to our setting, Memento (Zhou et al., 2025) learns a case-selection policy online from a scalar reward for the whole episode. LeRe combines both levers: its Planner renders each query into a structured key of skills and domain, and CCME learns to match it online from the Curator's per-entry attribution, without ground-truth labels or likelihoods. It thus runs on black-box models and credits individual entries, not whole episodes.

3 METHOD

3.1 PROBLEM SETUP AND NOTATION

We study inference-time adaptation of a *frozen* LLM: performance must improve over a stream of queries with no update to the base model. Let $\{x_i\}_{i=1}^{N}$ be queries drawn from an unknown distribution $\mathcal{D}_{\text{test}}$ and presented one at a time. For each x_i the system outputs a prediction $\hat{y}_i$ and a trajectory $\tau_i = (s_1, \ldots, s_T)$ of reasoning and tool-use steps, and is evaluated by the cumulative error $\sum_{i=1}^{N} \ell(\hat{y}_i, y_i^*)$. The labels y_i^* serve evaluation only and are never a training signal; supervision comes from implicit feedback such as execution outcomes and reflective judgment. Adaptation is carried by a memory bank $\mathcal{M}_{i-1} = \{m_1, m_2, \ldots\}$ of reusable entries (Section 3.2) and by two encoders: a frozen sentence encoder f composed with learned linear heads $W_q, W_m \in \mathbb{R}^{D\times D}$ and ℓ_2-normalized,

$$E_q(\cdot) = \frac{W_q\, f(\cdot)}{\|W_q\, f(\cdot)\|_2}, \qquad E_m(\cdot) = \frac{W_m\, f(\cdot)}{\|W_m\, f(\cdot)\|_2}, \tag{1}$$

so that $\mathrm{sim}(z, z') = z^\top z'$ is cosine similarity. The pair (E_q, E_m) is the *Contrastive Contextual Memory Encoder* (CCME); W_q, W_m, initialized to the identity, are the only trainable parameters. The base LLM acts in three roles under three prompts: a ***Planner*** P, a ***Solver*** G, and a ***Curator*** C.

Planner. P reads x_i without solving it and returns a probe, from which a fixed template ϕ builds the *retrieval key*:

$$\pi_i = (c_i,\, u_i,\, d_i,\, \mathcal{T}_i) \leftarrow P(x_i), \qquad k_i = \phi(\pi_i) = \text{“}c_i\ u_i \mid \text{domain: } d_i \mid \text{skills: } \mathcal{T}_i\text{”}. \tag{2}$$

Here c_i restates the question (*semantic context*), u_i describes any image in text (*visual context*), d_i is a domain label, and $\mathcal{T}_i$ a set of skill tags. The same ϕ renders memory entries (Section 3.2), so both towers embed strings of one form; we write $E_q(x_i) \triangleq E_q(k_i)$.

Retrieval. Each admissible entry $m \in \mathcal{C}_i \subseteq \mathcal{M}_{i-1}$, i.e., active, is scored by

$$\mathrm{score}(x_i, m) = \alpha\, \mathrm{sim}\big(E_q(x_i), E_m(m)\big) + (1-\alpha)\, \hat{p}(m), \qquad \alpha \in [0,1], \tag{3}$$

where $\hat{p}(m) \in (0,1)$ denotes the entry's empirical reliability (Eq. 8). The final subset $\mathcal{R}_i = \mathrm{MMR}_K(\mathcal{C}_i)$, with $|\mathcal{R}_i| \le K$, is selected greedily via Maximal Marginal Relevance (Carbonell & Goldstein, 1998) to balance relevance against intra-memory redundancy.

Solver. G conditions on the query and the retrieved entries, treated as advisory, $(\tau_i, \hat{y}_i) \leftarrow G(x_i, \mathcal{R}_i)$, and may execute code at most κ times within τ_i, each program's actual output being appended to the trajectory.

Curator. After $\hat{y}_i$ is committed, C observes the trajectory and the implicit feedback without observing the ground truth y_i^*, and returns

$$(v_i,\, a_i,\, \lambda_i,\, \mathcal{P}_i) \leftarrow C(x_i,\, \tau_i,\, \hat{y}_i,\, \mathcal{R}_i), \tag{4}$$

where $v_i = (o_i, \gamma_i)$ is a verification signal with verdict $o_i \in \{\text{correct}, \text{wrong}\}$ and confidence $\gamma_i \in [0,1]$, λ_i is a distilled lesson, $\mathcal{P}_i$ a set of proposed new entries, and a_i an attribution assigning every retrieved entry one label,

$$a_i : \mathcal{R}_i \to \{\textsc{Helpful},\ \textsc{Harmful},\ \textsc{Irrelevant},\ \textsc{Redundant}\}. \tag{5}$$

The bank then advances by

$$\mathcal{M}_i = \mathcal{M}_{i-1} \oplus \Delta\mathcal{M}_i, \tag{6}$$

where $\oplus$ screens each proposal in $\mathcal{P}_i$ and either rejects it, merges it into an existing entry, or writes it as a new one, and revises the reliability of existing entries from a_i and v_i; a_i also supplies the training pairs for CCME. Section 3.2 specifies both.

3.2 Memory Structure and Contrastive Memory Encoding

Entries. Each entry is a structured record

$$m = \big(\underbrace{\texttt{title},\ d,\ \mathcal{T}}_{\text{identity}},\ \underbrace{\texttt{bullets},\ \texttt{example}}_{\text{content}},\ \underbrace{\texttt{status},\ \texttt{meta}}_{\text{state}}\big), \tag{7}$$

whose *identity* names the skill (a short title, a domain d, and tags $\mathcal{T}$), whose *content* carries it out (bullets of actionable steps and an optional worked example), and whose *state* records its standing ($\texttt{status} \in \{\texttt{active}, \texttt{quarantined}\}$ and usage statistics $\texttt{meta}$). Only the identity is embedded for retrieval, mirroring the Planner's key k_i of Eq. 2 slot for slot.

Reliability. For each memory item m, the state $\texttt{meta}$ tracks confidence-weighted evidence of positive and negative utility, denoted $h_m, f_m \in \mathbb{R}_{\geq 0}$, indicating whether m helped or harmed downstream performance when retrieved (Eq. 12). Let $\theta_m \in [0,1]$ denote the latent probability that retrieving m contributes to a successful outcome. Placing a uniform prior $\theta_m \sim \mathrm{Beta}(1,1)$ and interpreting the accumulated evidence as fractional pseudo-observations yields the posterior $\mathrm{Beta}(1+h_m,\ 1+f_m)$. Its mean provides a closed-form, Laplace-smoothed reliability score:

$$\hat{p}(m) = \frac{h_m + 1}{h_m + f_m + 2} \ \in\ (0,1), \tag{8}$$

which instantiates the reliability component in Eq. 3. With no accumulated evidence ($h_m = f_m = 0$), $\hat{p}(m) = \frac{1}{2}$, the mean of the uniform prior. Being a deterministic function of the accumulated evidence, $\hat{p}(m)$ is recomputed in closed form on each memory write.

Update operator. $\oplus$ applies three passes to each proposal $m' \in \mathcal{P}_i$. (i) *Guard*: reject m' if it contains the reference answer, answer-revealing phrasing, or copies the question beyond an n-gram bound. (ii) *Consolidate* against the nearest same-domain entry,

$$m^\star = \operatorname*{arg\,max}_{m \in \mathcal{M}_{i-1}:\ d_m = d_{m'}} \mathrm{sim}\big(E_m(m'),\ E_m(m)\big), \tag{9}$$

merging into $m^\star$ (union of bullets, identity kept) if the similarity is at least θ_{merge}, creating a new entry linked to $m^\star$ if at least θ_{link}, and creating an unlinked entry otherwise. (iii) *Maintain*: quarantine any entry with $\hat{p}(m) < \rho$ and $h_m + f_m \geq e_{\min}$ (retained, no longer retrieved), and enforce a capacity $C_{\max}$ on the active set by pruning the weakest entries, by reliability and then by usage.

CCME training. The attribution of a scored item defines one training pair with anchor k_i,

$$\begin{aligned} \mathcal{S}_i^+ &= \{\, m \in \mathcal{R}_i : a_i(m) = \textsc{Helpful} \,\}, \\ \mathcal{S}_i^- &= \{\, m \in \mathcal{R}_i : a_i(m) \in \{\textsc{Harmful},\ \textsc{Irrelevant}\} \,\}, \end{aligned} \tag{10}$$

where Redundant entries (relevant but already covered) join neither set and a query with $\mathcal{S}_i^+ = \emptyset$ yields no pair. Writing $\sigma_i(m) \triangleq \mathrm{sim}(E_q(k_i), E_m(m))/\tau$ for a fixed temperature τ, a batch $\mathcal{B}$ of anchor–positive terms is trained by

$$\mathcal{L}_{\mathrm{CCME}} = -\frac{1}{|\mathcal{B}|} \sum_{(i,\, m^+) \in \mathcal{B}} \log \frac{e^{\sigma_i(m^+)}}{e^{\sigma_i(m^+)} + \sum_{m^- \in \mathcal{N}_i} e^{\sigma_i(m^-)}}, \qquad \mathcal{N}_i = \mathcal{S}_i^- \cup \bigcup_{j \neq i} \mathcal{S}_j^+, \tag{11}$$

where $\mathcal{N}_i$ joins the anchor's own hard negatives with in-batch negatives, the positives of the other anchors in the batch. Pairs enter a first-in-first-out buffer; every k_{upd} queries, once at least $n_{\min}$ trainable pairs are buffered, the heads take one AdamW step on Eq. 11 and all $E_m(m)$ are recomputed. The reliability $\hat{p}$ enters the score of Eq. 3 but not $\mathcal{L}_{\mathrm{CCME}}$.

3.3 The Agentic Loop

Figure 2 and Algorithm 1 compose these components into one *plan–retrieve–solve–curate* cycle per query. The memory bank, the projection heads, and their pair buffer are the only mutable state, reinitialized at each run ($\mathcal{M}_0 = \emptyset$, $W_q = W_m = I$); all reported gains are therefore acquired online from the evaluation stream, through memory updates and one training objective, $\mathcal{L}_{\mathrm{CCME}}$.

Credit assignment. Verdict and attribution jointly update the evidence of each retrieved entry,

$$(h_m,\, f_m)\ \mathrel{+}=\ \begin{cases}(\gamma_i,\, 0) & a_i(m) = \textsc{Helpful},\ o_i = \text{correct},\\ (0,\, \gamma_i) & a_i(m) = \textsc{Helpful},\ o_i = \text{wrong},\\ (0,\, \gamma_i) & a_i(m) = \textsc{Harmful},\ o_i = \text{wrong},\\ (0,\, 0) & \text{otherwise},\end{cases} \tag{12}$$

so credit and blame are symmetric in the confidence, while a HARMFUL report on a correct answer and any IRRELEVANT or REDUNDANT entry accrue none. Under execution-based verification, o_i compares $\hat{y}_i$ with the output of the program G ran; when none ran, C's own judgment supplies the verdict at capped confidence.

Order of operations. Retrieval reads a frozen snapshot of $\mathcal{M}_{i-1}$ and never writes. The evidence update, $\oplus$, and pair intake all occur after x_i has been scored, so no item's feedback reaches the bank or the heads that served it; the protocol is prequential.

Cost. A query makes at most $\kappa + 3$ LLM calls: P, C, and $1 + \kappa$ turns of G. Retrieval is an exact inner product, $\mathcal{O}(|\mathcal{M}|D)$ with $|\mathcal{M}| \le C_{\max}$; an encoder update costs $\mathcal{O}((|\mathcal{B}| + S)D^2)$ over S cached candidate views, once per k_{upd} queries. Per-query latency is dominated by the Solver.

Algorithm 1 Inference-time adaptation with a frozen LLM and CCME

Require: Frozen LLM in roles P, G, C; frozen base encoder f; stream $\{x_i\}_{i=1}^{N}$

$\mathcal{M}_0 \leftarrow \emptyset$; $W_q, W_m \leftarrow I$; buffer $\leftarrow \emptyset$
for $i = 1$ to N **do**
  $\pi_i \leftarrow P(x_i)$; $k_i \leftarrow \phi(\pi_i)$ ▷ plan, Eq. 2
  $\mathcal{R}_i \leftarrow \text{MMR}_K(\mathcal{C}_i)$ scored by Eq. 3 ▷ retrieve
  $(\tau_i, \hat{y}_i) \leftarrow G(x_i, \mathcal{R}_i)$ ▷ solve, $\le \kappa$ executions
  $(v_i, a_i, \lambda_i, \mathcal{P}_i) \leftarrow C(x_i, \tau_i, \hat{y}_i, \mathcal{R}_i)$ ▷ curate, Eq. 4
  update (h_m, f_m) for $m \in \mathcal{R}_i$ by Eq. 12
  $\mathcal{M}_i \leftarrow \mathcal{M}_{i-1} \oplus \Delta\mathcal{M}_i$ ▷ guard, consolidate, maintain
  if $\mathcal{S}_i^+ \neq \emptyset$ **then**
    buffer $\leftarrow$ buffer $\cup \{(k_i, \mathcal{S}_i^+, \mathcal{S}_i^-)\}$ ▷ Eq. 10
  end if
  if $i \bmod k_{\text{upd}} = 0$ **and** buffer holds $\ge n_{\min}$ trainable pairs **then**
    $(W_q, W_m) \leftarrow$ AdamW step on $\mathcal{L}_{\text{CCME}}$ ▷ Eq. 11
    recompute $E_m(m)$ for all $m \in \mathcal{M}_i$
  end if
end for

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate LeRe on twelve streams spanning three reasoning regimes. *Mathematics*: AIME 2024 (HuggingFaceH4, 2024), AIME 2025 (MathArena, 2025), the pooled AIME 2020–2025, and MATH (Hendrycks et al., 2021). *Scientific knowledge*: GPQA-Diamond (Rein et al., 2024), the Engineering and Physics subsets of MMLU-Pro (Wang et al., 2024b), and HLE-Exact, a subset of Humanity's Last Exam (Phan et al., 2025) with exactly scoreable answers that probes frontier difficulty. *Multimodal reasoning*: MathVista (Lu et al., 2023) and the three MMMU-Pro (Yue et al., 2025) splits (Standard with four and with ten options, and Vision). Each stream is processed once, in a fixed order shared by all methods, and scored by Pass@1.

Baselines. We compare against a zero-shot model without memory (Baseline), retrieval-augmented in-context learning over the model's own past solutions (RAG-ICL) (Agarwal et al., 2024), the Cumulative and Retrieval-Synthesis variants of Dynamic Cheatsheet (DC Cu, DC RS) (Suzgun et al., 2026), and Agentic Context Engineering (ACE) (Zhang et al., 2025). Like LeRe, every memory-based method adapts online on the same stream without ground-truth labels.

Backbone Models. Our main evaluation uses three widely deployed models at different capability levels: Gemini-3.1-flash-lite (Google, 2026), GPT-4.1-mini (OpenAI, 2025), and GPT-4o-mini (OpenAI, 2024), each frozen and decoded at temperature 0. Detailed descriptions of the

Table 1: **Main benchmark evaluation across twelve reasoning streams.** Pass@1 accuracy (%) on mathematical, scientific, and multimodal reasoning tasks; HLE denotes the HLE-Exact subset. Baseline (BL) denotes direct zero-shot inference without memory. For each method, we report the absolute gain over Baseline in percentage points (Δ = Method − BL). **Average** denotes the arithmetic mean gain across all twelve evaluation streams. **Green** and **red** gains indicate positive and negative absolute gains over Baseline, respectively; (0.0) indicates a tie. **Bold** denotes the best result in each column within a model block.

Gemini-3.1-flash-lite

Method	Mathematical reasoning				Scientific knowledge				Multimodal reasoning				Average
	AIME 2024	AIME 2025	AIME 2020-2025	MATH	GPQA	MMLU Eng.	MMLU Physics.	HLE	Math Vista	MMMU-Pro Std. 4	MMMU-Pro Std. 10	MMMU-Pro Vision	
Baseline	56.7	30.0	45.1	94.0	73.2	77.6	86.4	9.6	91.2	**83.2**	75.2	72.8	
RAG-ICL	60.0 (+3.3 ↑)	33.3 (+3.3 ↑)	42.0 (-3.1 ↓)	92.4 (-1.6 ↓)	73.2 (0.0)	77.2 (-0.4 ↓)	89.2 (+2.8 ↑)	9.2 (-0.4 ↓)	90.8 (-0.4 ↓)	80.8 (-2.4 ↓)	73.2 (-2.0 ↓)	71.2 (-1.6 ↓)	-0.2 ↓
DC Cu	63.3 (+6.6 ↑)	46.7 (+16.7 ↑)	54.3 (+9.2 ↑)	93.6 (-0.4 ↓)	69.7 (-3.5 ↓)	78.0 (+0.4 ↑)	88.8 (+2.4 ↑)	11.2 (+1.6 ↑)	92.0 (+0.8 ↑)	**83.2 (0.0)**	**76.8 (+1.6 ↑)**	72.4 (-0.4 ↓)	+2.9 ↑
DC RS	66.7 (+10.0 ↑)	36.7 (+6.7 ↑)	59.3 (+14.2 ↑)	93.6 (-0.4 ↓)	**74.7 (+1.5 ↑)**	**81.2 (+3.6 ↑)**	**90.0 (+3.6 ↑)**	10.0 (+0.4 ↑)	**92.4 (+1.2 ↑)**	**83.2 (0.0)**	**76.8 (+1.6 ↑)**	**75.2 (+2.4 ↑)**	+3.7 ↑
ACE	56.7 (0.0)	30.0 (0.0)	43.2 (-1.9 ↓)	**94.8 (+0.8 ↑)**	72.2 (-1.0 ↓)	77.2 (-0.4 ↓)	88.8 (+2.4 ↑)	8.0 (-1.6 ↓)	90.8 (-0.4 ↓)	81.6 (-1.6 ↓)	74.4 (-0.8 ↓)	72.8 (0.0)	-0.4 ↓
LeRe	**76.7 (+20.0 ↑)**	**60.0 (+30.0 ↑)**	**69.8 (+24.7 ↑)**	94.0 (0.0)	**74.7 (+1.5 ↑)**	78.0 (+0.4 ↑)	**90.0 (+3.6 ↑)**	**12.4 (+2.8 ↑)**	91.2 (0.0)	82.0 (-1.2 ↓)	**76.8 (+1.6 ↑)**	72.8 (0.0)	**+6.9 ↑**

GPT-4.1-mini

Method	Mathematical reasoning				Scientific knowledge				Multimodal reasoning				Average
	AIME 2024	AIME 2025	AIME 2020-2025	MATH	GPQA	MMLU Eng.	MMLU Physics.	HLE	Math Vista	MMMU-Pro Std. 4	MMMU-Pro Std. 10	MMMU-Pro Vision	
Baseline	33.3	23.3	24.1	84.0	**68.2**	57.6	82.4	4.8	77.2	72.8	61.2	51.2	
RAG-ICL	36.7 (+3.4 ↑)	23.3 (0.0)	16.0 (-8.1 ↓)	82.8 (-1.2 ↓)	57.6 (-10.6 ↓)	51.6 (-6.0 ↓)	80.8 (-1.6 ↓)	4.4 (-0.4 ↓)	76.4 (-0.8 ↓)	66.8 (-6.0 ↓)	60.4 (-0.8 ↓)	56.0 (+4.8 ↑)	-2.3 ↓
DC Cu	36.7 (+3.4 ↑)	36.7 (+13.4 ↑)	28.4 (+4.3 ↑)	80.0 (-4.0 ↓)	63.1 (-5.1 ↓)	55.6 (-2.0 ↓)	78.4 (-4.0 ↓)	6.4 (+1.6 ↑)	73.6 (-3.6 ↓)	70.8 (-2.0 ↓)	61.6 (+0.4 ↑)	58.0 (+6.8 ↑)	+0.8 ↑
DC RS	46.7 (+13.4 ↑)	33.3 (+10.0 ↑)	35.8 (+11.7 ↑)	83.2 (-0.8 ↓)	64.1 (-4.1 ↓)	59.2 (+1.6 ↑)	77.6 (-4.8 ↓)	5.6 (+0.8 ↑)	76.8 (-0.4 ↓)	72.0 (-0.8 ↓)	57.2 (-4.0 ↓)	56.8 (+5.6 ↑)	+2.4 ↑
ACE	36.7 (+3.4 ↑)	30.0 (+6.7 ↑)	27.2 (+3.1 ↑)	63.2 (-20.8 ↓)	64.6 (-3.6 ↓)	56.4 (-1.2 ↓)	**84.0 (+1.6 ↑)**	4.0 (-0.8 ↓)	75.2 (-2.0 ↓)	70.4 (-2.4 ↓)	59.2 (-2.0 ↓)	54.0 (+2.8 ↑)	-1.3 ↓
LeRe	**60.0 (+26.7 ↑)**	**56.7 (+33.4 ↑)**	**45.1 (+21.0 ↑)**	**88.4 (+4.4 ↑)**	66.7 (-1.5 ↓)	**64.0 (+6.4 ↑)**	81.2 (-1.2 ↓)	**8.4 (+3.6 ↑)**	**79.6 (+2.4 ↑)**	**75.2 (+2.4 ↑)**	**63.2 (+2.0 ↑)**	**59.6 (+8.4 ↑)**	**+9.0 ↑**

GPT-4o-mini

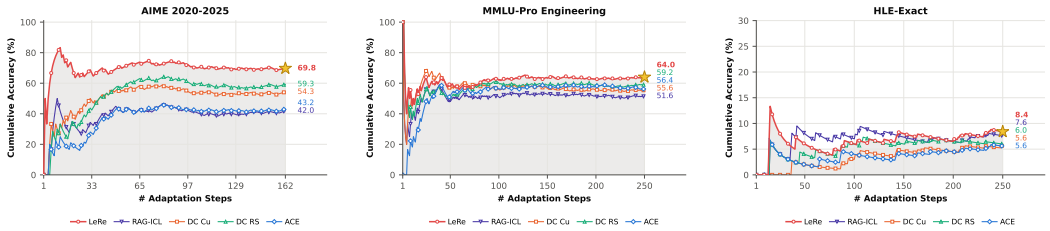
Method	Mathematical reasoning				Scientific knowledge				Multimodal reasoning				Average
	AIME 2024	AIME 2025	AIME 2020-2025	MATH	GPQA	MMLU Eng.	MMLU Physics.	HLE	Math Vista	MMMU-Pro Std. 4	MMMU-Pro Std. 10	MMMU-Pro Vision	
Baseline	16.7	3.3	7.4	**72.8**	34.3	**37.6**	55.2	6.8	57.6	52.4	37.6	26.4	
RAG-ICL	10.0 (-6.7 ↓)	3.3 (0.0)	7.4 (0.0)	69.6 (-3.2 ↓)	34.3 (0.0)	29.2 (-8.4 ↓)	39.2 (-16.0 ↓)	7.6 (+0.8 ↑)	61.2 (+3.6 ↑)	50.0 (-2.4 ↓)	41.2 (+3.6 ↑)	28.8 (+2.4 ↑)	-2.2 ↓
DC Cu	13.3 (-3.4 ↓)	13.3 (+10.0 ↑)	14.8 (+7.4 ↑)	70.8 (-2.0 ↓)	33.8 (-0.5 ↓)	32.4 (-5.2 ↓)	54.8 (-0.4 ↓)	5.6 (-1.2 ↓)	58.4 (+0.8 ↑)	**56.4 (+4.0 ↑)**	**44.4 (+6.8 ↑)**	34.8 (+8.4 ↑)	+2.1 ↑
DC RS	13.3 (-3.4 ↓)	16.7 (+13.4 ↑)	19.1 (+11.7 ↑)	68.4 (-4.4 ↓)	32.3 (-2.0 ↓)	34.0 (-3.6 ↓)	**58.4 (+3.2 ↑)**	6.0 (-0.8 ↓)	62.0 (+4.4 ↑)	56.0 (+3.6 ↑)	**44.4 (+6.8 ↑)**	**36.4 (+10.0 ↑)**	+3.2 ↑
ACE	16.7 (0.0)	3.3 (0.0)	8.6 (+1.2 ↑)	71.2 (-1.6 ↓)	35.9 (+1.6 ↑)	34.8 (-2.8 ↓)	57.6 (+2.4 ↑)	5.6 (-1.2 ↓)	**62.4 (+4.8 ↑)**	54.4 (+2.0 ↑)	38.0 (+0.4 ↑)	30.4 (+4.0 ↑)	+0.9 ↑
LeRe	**26.7 (+10.0 ↑)**	**20.0 (+16.7 ↑)**	**21.6 (+14.2 ↑)**	67.2 (-5.6 ↓)	**40.9 (+6.6 ↑)**	31.6 (-6.0 ↓)	54.0 (-1.2 ↓)	**8.4 (+1.6 ↑)**	60.8 (+3.2 ↑)	55.2 (+2.8 ↑)	41.2 (+3.6 ↑)	35.6 (+9.2 ↑)	**+4.6 ↑**

datasets (Appendix B.1) and baseline methods (Appendix B.2), together with the model settings, online-training hyperparameters, and the order of each test stream, are given in Appendix B.

4.2 Main Results

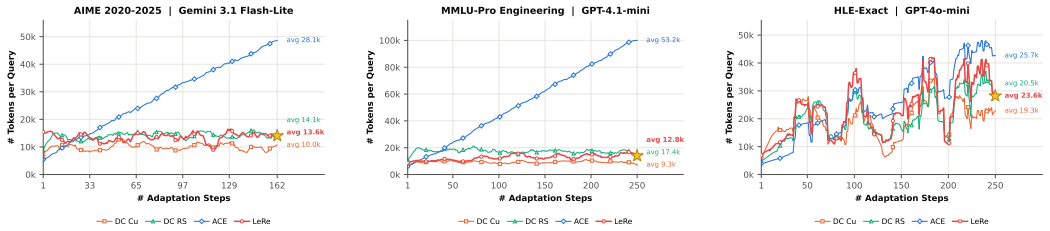
Overall performance. Table 1 shows that LeRe attains the largest average gain over the no-memory baseline on every backbone, $+6.9\%$, $+9.0\%$, and $+4.6\%$ on Gemini-3.1-flash-lite, GPT-4.1-mini, and GPT-4o-mini, exceeding the strongest competitor, DC RS ($+3.7\%$, $+2.4\%$, $+3.2\%$), by 1.4% to 6.6%, while ACE lowers average accuracy on two of the three backbones. The gains are largest where reasoning is hardest. LeRe is best in all nine AIME cells, and on AIME 2020–2025 it exceeds the baseline by 24.7%, 21.0%, and 14.2% and the strongest competitor by 10.5%, 9.3%, and 2.5%. On HLE-Exact, the hardest stream for every backbone, LeRe is also best on all three, although by a small margin (0.8% to 2.0%). Beyond these shared trends, how LeRe helps depends on the backbone's own capability, which we proxy by its mean baseline accuracy over the twelve streams. For GPT-4.1-mini (53.3%), a capable model that still leaves headroom on most streams, the benefit is broadest: LeRe is best on 10 of 12 streams across all three regimes, including MMLU-Pro Engineering (64.0%, $+6.4\%$), where three of the four competing methods fall below the baseline. For the strongest backbone, Gemini-3.1-flash-lite (66.3%), the gain concentrates where headroom remains: AIME accounts for 90% of it, while on the other nine streams, several near saturation (*e.g.*, 94.0% on MATH), LeRe changes accuracy by $+1.0\%$ on average. For the weakest backbone, GPT-4o-mini (34.0%), LeRe helps selectively, with its largest gains outside AIME on MMMU-Pro Vision ($+9.2\%$) and GPQA-Diamond ($+6.6\%$), but trails the baseline on MATH (-5.6%) and MMLU-Pro Engineering (-6.0%). This pattern is consistent with a memory whose value is bounded by the model's ability to curate reliable entries and act on them, although with three backbones it is suggestive rather than conclusive.

Adaptation dynamics analysis. Figure 3 tracks cumulative accuracy over each stream. After the noisy first steps, LeRe pulls ahead and stays level with or ahead of every method to the end, from step 1 on AIME 2020–2025 with Gemini-3.1-flash-lite, step 74 on MMLU-Pro Engineering with GPT-4.1-mini, and step 143 on HLE-Exact with GPT-4o-mini. Its accuracy over the final third of each stream matches or exceeds that over the first third (70.4% vs. 70.4%, 64.3% vs. 61.4%, and 9.5% vs. 4.8%), so the gains accumulate rather than fade as memory grows. Figure 4 shows what each method pays per query for this adaptation. ACE injects its full playbook at every step, so its prompt grows almost linearly, to about 49k tokens per query by the end of AIME and 100k by



(a) Gemini-3.1-flash-lite (b) GPT-4.1-mini (c) GPT-4o-mini

Figure 3: **Online adaptation over the evaluation stream.** Cumulative Pass@1 accuracy (%) as each stream is processed in order, for (a) Gemini-3.1-flash-lite on AIME 2020–2025, (b) GPT-4.1-mini on MMLU-Pro Engineering, and (c) GPT-4o-mini on HLE-Exact; stars (★) mark LeRe's final accuracy. LeRe pulls ahead in every stream and does not degrade late in the stream.

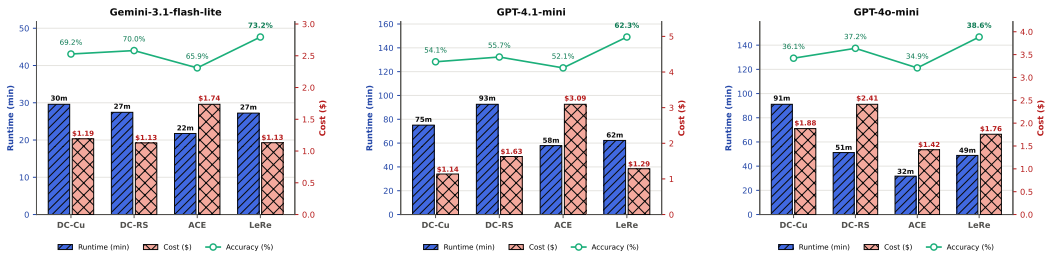


(a) Gemini-3.1-flash-lite (b) GPT-4.1-mini (c) GPT-4o-mini

Figure 4: **Token consumption over the evaluation stream.** Total LLM tokens per query (moving average) for the same streams as Figure 3 for (a) Gemini-3.1-flash-lite on AIME 2020–2025, (b) GPT-4.1-mini on MMLU-Pro Engineering, and (c) GPT-4o-mini on HLE-Exact; stars (★) mark LeRe's final consumption. ACE grows almost linearly with the stream, while LeRe stays flat as its memory grows.

the end of MMLU-Pro Engineering, yet it ends below the no-memory baseline on both (43.2% vs. 45.1% and 56.4% vs. 57.6%): memory that is injected rather than selected adds cost without adding accuracy. LeRe's consumption instead stays flat, since top-K retrieval bounds the memory in each prompt however large the store becomes. On these two streams it uses fewer tokens per query than DC RS (13.6k vs. 14.1k and 12.8k vs. 17.4k) and 52% and 76% fewer than ACE, while exceeding DC RS by 10.5% and 4.8%. On HLE-Exact, consumption varies with the query for every method, and LeRe (23.6k) sits between DC and ACE. LeRe's gains thus come from which entries enter the prompt, not from how many.

Cost and speed analysis. Figure 5 compares the memory-curating methods on mean runtime and API cost per stream, together with accuracy averaged over all 12 streams. LeRe is the most accurate on every backbone at a comparable cost. With Gemini-3.1-flash-lite, it ties DC RS as the cheapest method ($1.13 per stream) while being 3.2% more accurate. With GPT-4.1-mini, it is the second cheapest ($1.29, against $1.14 for DC Cu) and 6.6% ahead of the next-best method. With GPT-4o-mini, it costs less than both DC variants but more than ACE ($1.76 vs. $1.42). ACE is the fastest method on every backbone, but it pays for its full playbook in every prompt: it is the most expensive on two backbones, at 1.5× and 2.4× LeRe's cost, and the least accurate of the four methods on all three. LeRe's runtime is 1.1 to 1.5 times that of ACE, because it issues 3.2 to 4.3 LLM calls per query on average (Planner, Solver turns, and Curator) and executes code, yet it is faster than both DC variants with the GPT backbones and on par with them with Gemini. These extra calls stay cheap because each carries only the top-K entries rather than the whole memory. As ACE's prompt grows with the stream while LeRe's does not (Figure 4), the cost comparison should shift further in LeRe's favor on streams longer than those evaluated here.



(a) Gemini-3.1-flash-lite (b) GPT-4.1-mini (c) GPT-4o-mini

Figure 5: **Inference runtime, cost, and accuracy.** Mean runtime per stream in minutes (blue, left axis) and mean API cost per stream in USD at list prices (red, right axis), shown with Pass@1 accuracy averaged over the 12 streams of Table 1 (green), for (a) Gemini-3.1-flash-lite, (b) GPT-4.1-mini, and (c) GPT-4o-mini. Runtime and cost are averaged over the streams on which every method has a record. LeRe is the most accurate method on every backbone at a cost close to the cheapest.

Table 2: **Ablation of LeRe across representative benchmarks.** Accuracy (%) on nine model–stream pairs when a single component is removed and all others are kept fixed: *w/o Planner* keys retrieval on the raw question, *w/o CCME* retrieves with the frozen sentence encoder, and *w/o Execution* disables code execution by the Solver. Values in parentheses are absolute changes relative to the full LeRe; **Average** is their unweighted mean over the nine pairs.

Method	Gemini-3.1-flash-lite			GPT-4.1-mini			GPT-4o-mini			Average
	AIME 2020-2025	MMMU-Pro Std. 10	HLE	AIME 2020-2025	MMMU-Pro Vision	MMLU Eng.	AIME 2020-2025	MMMU-Pro Std. 4	GPQA	
w/o Planner	65.4 (-4.4 ↓)	70.0 (-6.8 ↓)	10.8 (-1.6 ↓)	38.9 (-6.2 ↓)	58.0 (-1.6 ↓)	59.6 (-4.4 ↓)	19.1 (-2.5 ↓)	52.8 (-2.4 ↓)	36.4 (-4.5 ↓)	**-3.8** ↓
w/o CCME	66.7 (-3.1 ↓)	74.0 (-2.8 ↓)	10.4 (-2.0 ↓)	38.3 (-6.8 ↓)	56.4 (-3.2 ↓)	61.6 (-2.4 ↓)	20.8 (-0.8 ↓)	53.6 (-1.6 ↓)	34.8 (-6.1 ↓)	**-3.2** ↓
w/o Execution	43.8 (-26.0 ↓)	72.4 (-4.4 ↓)	9.2 (-3.2 ↓)	31.5 (-13.6 ↓)	52.4 (-7.2 ↓)	58.8 (-5.2 ↓)	4.9 (-16.7 ↓)	49.2 (-6.0 ↓)	37.9 (-3.0 ↓)	**-9.5** ↓
LeRe (full)	**69.8**	**76.8**	**12.4**	**45.1**	**59.6**	**64.0**	**21.6**	**55.2**	**40.9**	

4.3 Ablation Study

Table 2 removes one component at a time on nine model–stream pairs. Every removal lowers accuracy on every pair, on average by 3.8% without the Planner, 3.2% without CCME, and 9.5% without code execution, the largest drop. Removing execution hurts most on AIME 2020–2025 (-26.0%, -13.6%, -16.7%), where LeRe falls below the no-memory baseline on two backbones (43.8% vs. 45.1% and 4.9% vs. 7.4%). It also costs 3.0% to 7.2% elsewhere, even on MMLU-Pro Engineering, where code runs on only 8% of problems. The AIME losses arise because code computes exactly, more reliably than chain-of-thought (Suzgun et al., 2026). The losses where code rarely runs arise from verification: without execution, credit rests on the Curator's judgment, which can grow optimistic, accepting wrong answers twice as often on Gemini AIME (25% vs. 13%). The Planner and CCME have smaller but consistent effects. The Planner matters most on MMMU-Pro Standard-10 (-6.8%), where its image description is the only visual information in the retrieval key, and GPT-4.1-mini AIME (-6.2%). CCME's effect tracks its training signal: removing it costs 6.1% on GPQA-Diamond, where its heads were updated 27 times, but only 0.8%, within run-to-run variance, on GPT-4o-mini AIME, where they were not updated.

5 Conclusion

We presented LeRe, a self-improving agentic framework in which a frozen LLM plans, solves, and curates, learning which memory to retrieve rather than how much to store. CCME, trained online from the Curator's attribution, lets retrieval favor what has helped, while the Planner sharpens each retrieval key and the GCM gate keeps memory selective. LeRe learns without ground-truth labels or weight updates. Across twelve streams and three backbones, it attains the highest average accuracy at near-lowest cost, its gains accumulate over each stream, and its per-query token cost stays flat while full-memory injection grows. Its limits are a learning signal only as reliable as its verification, weakest where no code can check an answer, and a multi-call loop adding runtime. LeRe's consistently superior performance points to a scalable, efficient, and effective way to support inference-time adaptation in LLMs.

AI USE STATEMENT

We used LLMs only as a writing aid during the preparation of this submission, to polish the clarity and readability of the text (e.g., refining sentence structure, correcting grammar, and shortening verbose phrasing). All research ideas, the design of the method, the experiments, and the analyses were conceived, implemented, and validated by the authors, who reviewed all AI-assisted text and take full responsibility for the content of this paper.

ETHICS STATEMENT

This work does not raise specific ethical concerns. Our contributions focus on developing an algorithmic framework for inference-time memory in large language models (LLMs). All experiments are conducted on publicly available benchmarks with commercially available models accessed through their public APIs, without involving human subjects, sensitive data, or privacy-related information. No potential conflicts of interest are present.

REPRODUCIBILITY STATEMENT

Appendix B.1 describes how every evaluation stream is sampled and ordered (a fixed seed-10 permutation shared by all methods), and Appendix B.3 gives the evaluation protocol and scoring rules. Appendix B lists the backbone versions (Table 3), the run configuration of every method, and all of LeRe's hyperparameters (Table 4). The prompts are in Appendix D.1, and the assumptions and complete proofs of our theoretical results are in Appendix A. All methods decode at temperature 0, although outputs of the proprietary backbones may vary slightly across provider versions. Our code, including prompts, configurations, and evaluation scripts, is available at `https://anonymous.4open.science/r/learn-to-remember-AD37`.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Rishabh Agarwal, Avi Singh, Lei Zhang, Bernd Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, et al. Many-shot in-context learning. *Advances in Neural Information Processing Systems*, 37:76930–76966, 2024.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pp. 9112–9141, 2024.

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. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 335–336, 1998.

Ching-Yao Chuang, Joshua Robinson, Yen-Chen Lin, Antonio Torralba, and Stefanie Jegelka. Debiased contrastive learning. In *Advances in Neural Information Processing Systems*, 2020.

A. Philip Dawid. Present position and potential developments: Some personal views. statistical theory: The prequential approach. *Journal of the Royal Statistical Society. Series A (General)*, 147(2):278–292, 1984.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

Google. Gemini 3.1 flash-lite. Gemini API documentation, 2026. URL https://ai.google.dev/gemini-api/docs/models/gemini-3.1-flash-lite. Accessed 2026-09-17.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 297–304, 2010.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

HuggingFaceH4. AIME 2024. https://huggingface.co/datasets/HuggingFaceH4/aime_2024, 2024.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 6769–6781, 2020.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Zhuang Ma and Michael Collins. Noise contrastive estimation and negative sampling for conditional models: Consistency and statistical efficiency. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3698–3707, 2018.

MathArena. AIME 2025. https://huggingface.co/datasets/MathArena/aime_2025, 2025.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *International conference on machine learning*, pp. 16888–16905. PMLR, 2022.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

OpenAI. New embedding models and API updates. https://openai.com/index/new-embedding-models-and-api-updates/, January 2024.

OpenAI. GPT-4o mini: Advancing cost-efficient intelligence. Technical report, OpenAI, July 2024. URL https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/. Accessed: 2024.

OpenAI. Introducing GPT-4.1 in the API. OpenAI blog, 2025. URL https://openai.com/index/gpt-4-1/. Accessed 2026-09-17.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021. URL https://arxiv.org/abs/2103.00020.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 3982–3992, 2019.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. In *International Conference on Learning Representations*, 2021.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. Learning to retrieve prompts for in-context learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2022.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. REPLUG: Retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 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.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. MPNet: Masked and permuted pre-training for language understanding. In *Advances in Neural Information Processing Systems*, 2020.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory – code repository. https://github.com/suzgunmirac/dynamic-cheatsheet, 2025.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations (ICLR)*, 2021.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 2609–2634, 2023.

Liang Wang, Nan Yang, and Furu Wei. Learning to retrieve in-context examples for large language models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2024a.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, 2020.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024c.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2025.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. In *Advances in Neural Information Processing Systems*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, et al. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15134–15186, 2025.

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

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, and Jun Wang. Memento: Fine-tuning LLM agents without fine-tuning LLMs. *arXiv preprint arXiv:2508.16153*, 2025.

Yuxin Zuo, Kaiyan Zhang, Shang Qu, Li Sheng, Xuekai Zhu, Biqing Qi, Youbang Sun, Ganqu Cui, Ning Ding, and Bowen Zhou. TTRL: Test-time reinforcement learning. In *Advances in Neural Information Processing Systems*, 2025.

CONTENTS OF APPENDIX

A Theoretical Analysis **15**

A.1 Setup . . . 15

A.2 CCME Learns a Utility-Ordered Density Ratio . . . 15

A.3 Robustness to Noisy Attribution . . . 16

A.4 From Population to Finite Samples . . . 16

A.5 Reliability and Quarantine . . . 17

A.6 From Retrieval to Task Risk . . . 18

A.7 Bounded Context as Memory Grows. . . . 18

A.8 Validity of Learning on the Evaluation Stream . . . 19

A.9 Assumptions and Scope. . . . 19

B Details for Experiment Settings **20**

B.1 Datasets . . . 20

B.2 Benchmark Methods . . . 20

B.3 Evaluation Protocol . . . 20

B.4 Implementation Details . . . 21

C Additional Analyses **23**

C.1 Extended Results . . . 23

C.2 Mechanism Evidence . . . 24

C.3 Sensitivity Analysis . . . 26

D Prompts and Examples **28**

D.1 Prompt Templates . . . 28

D.2 Highly Influential Memory Entries. . . . 36

E Limitations and Future Directions **38**

E.1 Limitations. . . . 38

E.2 Future directions . . . 38

A THEORETICAL ANALYSIS

This section analyzes the learning principles behind LeRe's design. A complete characterization of the agentic system would require a model of how a frozen LLM responds to arbitrary prompts, tools, and memory, which is beyond reach. We therefore isolate the components whose behavior can be stated precisely and ask, for each, what it provably does under explicit assumptions: the CCME objective (§A.2–§A.4), the reliability estimate and quarantine rule (§A.5), the link from retrieval to task risk (§A.6), the cost of memory as it grows (§A.7), and the validity of learning online on the evaluation stream (§A.8). §A.9 states the limits of the analysis.

A.1 SETUP

Queries $x_1, \ldots, x_N$ arrive one at a time and are answered by a frozen LLM. For a query x and a memory entry m, let the *utility* of m for x be

$$U(x, m) = \mathbb{E}\big[\ell(\hat{y}_0(x), y^*) - \ell(\hat{y}(x, m), y^*) \,\big|\, x, m\big] \in [-1, 1], \tag{13}$$

where $\hat{y}_0(x)$ is the answer without memory, $\hat{y}(x, m)$ the answer with m in context, and $\ell \in [0, 1]$ a bounded loss. Utility is never observed; LeRe observes only the Curator's attribution $a(x, m) \in \{\text{HELPFUL}, \text{HARMFUL}, \text{IRRELEVANT}, \text{REDUNDANT}\}$ and its verdict $o \in \{\text{correct}, \text{wrong}\}$ with confidence $\gamma \in [0, 1]$.

Write $q = q(x)$ for the Planner's key (k_i in Eq. 2) and $v(m)$ for the rendered identity of m. CCME defines the learned similarity

$$g_\theta(x, m) = E_q(q)^\top E_m(v(m)), \qquad \theta = (W_q, W_m), \tag{14}$$

with E_q, E_m as in Eq. 1. The implementation also multiplies the similarity term of Eq. 3 by a domain weight $w \in [1 - \beta, 1]$ that discounts entries outside the query's domain (Table 4). We analyze candidates within one domain, where $w = 1$, so the score is

$$s(x, m) = \alpha\, g_\theta(x, m) + (1 - \alpha)\, \hat{p}(m), \qquad \alpha \in [0, 1]. \tag{15}$$

A.2 CCME LEARNS A UTILITY-ORDERED DENSITY RATIO

For an anchor x, CCME contrasts a positive $m^+ \sim p_+(\cdot \mid x)$, an entry the Curator labelled HELPFUL, against negatives $m^- \sim q_x$. In LeRe, q_x is a mixture of two sources (Eq. 10–11): *hard* negatives, entries retrieved for x and labelled HARMFUL or IRRELEVANT, and *in-batch* negatives, the positives of other anchors. The InfoNCE loss $\mathcal{L}_{\text{CCME}}$ is the cross-entropy of identifying m^+ among $\{m^+\} \cup \mathcal{N}$ with logits g_θ/τ.

Proposition A.1 (Population optimum; Gutmann & Hyvärinen, 2010; Oord et al., 2018; Ma & Collins, 2018)**.** *Over an unrestricted class of scores, any population minimizer of the InfoNCE loss satisfies, for each anchor x and every m in the support of q_x,*

$$g^\star(x, m) = \tau \log \frac{p_+(m \mid x)}{q_x(m)} + c(x), \tag{16}$$

where $c(x)$ does not depend on m.

Proof. Fix x and a candidate set of size $J + 1$ containing one positive. The Bayes posterior that candidate j is the positive is $\frac{p_+(m_j|x)/q_x(m_j)}{\sum_l p_+(m_l|x)/q_x(m_l)}$. The softmax cross-entropy is minimized, as a proper scoring rule, exactly when the model's softmax equals this posterior, which holds if and only if $g(x, m)/\tau$ equals the log ratio up to an additive term constant in m. Consistency of this estimator for conditional models with $J \geq 1$ negatives is shown by Ma & Collins (2018). □

The ratio is taken against q_x, not a fixed background distribution, so what CCME learns depends on how negatives are chosen. The next result states the condition under which it is ordered by utility.

Theorem A.2 (Utility-ordered ranking)**.** *Let μ be a base measure over entries and suppose that, for each x,*

$$p_+(m \mid x) \propto w_x\big(U(x, m)\big)\, \mu(m), \qquad q_x(m) \propto v_x\big(U(x, m)\big)\, \mu(m), \tag{17}$$

with w_x strictly increasing and $v_x > 0$ non-increasing. Then $g^\star(x, \cdot)$ is strictly increasing in $U(x, \cdot)$: $U(x, m_1) > U(x, m_2) \Rightarrow g^\star(x, m_1) > g^\star(x, m_2)$.

Proof. By Proposition A.1, $g^\star(x,m) = \tau \log w_x(U(x,m)) - \tau \log v_x(U(x,m)) + c'(x)$. The first term is strictly increasing and the second non-decreasing in U, so their sum is strictly increasing. □

Theorem A.2 explains two design choices. First, hard negatives are entries the Curator judged harmful or irrelevant, so their sampling weight v_x falls with utility, as the theorem requires. Second, it shows why REDUNDANT entries must not be negatives: they are relevant to x but already covered, so $U(x,m)$ is high, and placing them in q_x makes v_x increase on the high-utility region.

Remark A.3 (False negatives break the ordering). If v_x is not monotone, the ratio w_x/v_x can reverse the utility order: for $U(x,m_1) > U(x,m_2)$ it suffices that $v_x(U(x,m_1))/v_x(U(x,m_2)) > w_x(U(x,m_1))/w_x(U(x,m_2))$. Treating relevant-but-covered entries as negatives is an instance of the false-negative bias of contrastive learning (Chuang et al., 2020); excluding REDUNDANT entries removes it. In-batch negatives, drawn from other anchors' positives, keep q_x from collapsing onto the few hard negatives a top-K list supplies, which otherwise concentrates the gradient on a narrow region of the space (Robinson et al., 2021). They can themselves be false negatives when two anchors share a skill; this risk is small when anchors in a batch come from different problems, and it is the price of a well-anchored geometry.

A.3 ROBUSTNESS TO NOISY ATTRIBUTION

The Curator's labels are not ground truth. Model its errors as a mixture: the positive for x is drawn from $\tilde{p}_+ = (1-\eta)\, p_+ + \eta\, r_x$, where $\eta \in [0,1)$ is the rate of mislabelled positives and r_x their distribution.

Proposition A.4 (Noise that resembles negatives preserves the ranking). *Let $\rho(x,m) = p_+(m \mid x)/q_x(m)$ be the clean ratio and $\tilde{g}^\star$ the optimum under $\tilde{p}_+$.*

1. *If $r_x = q_x$, then $\tilde{g}^\star(x,m) = \tau \log\big((1-\eta)\,\rho(x,m) + \eta\big) + c(x)$, a strictly increasing function of ρ; the ranking of Theorem A.2 is preserved, and log-ratio gaps are contracted locally by the factor $(1-\eta)\rho/((1-\eta)\rho+\eta) < 1$.*
2. *If r_x concentrates on a lower-utility entry m_2 relative to a higher-utility m_1 with $q_x(m_1) = q_x(m_2)$, so that $\eta\,\big(r_x(m_2) - r_x(m_1)\big) > (1-\eta)\big(p_+(m_1 \mid x) - p_+(m_2 \mid x)\big)$, the two are ranked in reverse.*

Proof. Apply Proposition A.1 to $\tilde{p}_+$. In case 1, $\tilde{p}_+/q_x = (1-\eta)\rho + \eta$, which is strictly increasing in ρ; the derivative of $\log((1-\eta)\rho+\eta)$ with respect to $\log\rho$ is $(1-\eta)\rho/((1-\eta)\rho+\eta)$. In case 2, with equal q_x, the ordering of $\tilde{g}^\star$ follows that of $\tilde{p}_+$, and $\tilde{p}_+(m_2 \mid x) > \tilde{p}_+(m_1 \mid x)$ rearranges to the stated inequality. □

Unsystematic attribution errors therefore cost statistical efficiency, since margins shrink and more pairs are needed, but not correctness. Systematic errors, where the Curator repeatedly credits the same unhelpful entries, can reorder retrieval; this is the failure mode that the reliability term (§A.5) and an execution-anchored verdict are designed to limit.

A.4 FROM POPULATION TO FINITE SAMPLES

Let $\hat{s}_n$ be the score after n updates and $s^\star$ its population counterpart, and suppose

$$\sup_{x,m} |\hat{s}_n(x,m) - s^\star(x,m)| \le \varepsilon_n, \tag{18}$$

where ε_n absorbs estimation error, optimization error, and the approximation error of the linear heads.

Proposition A.5 (Pairwise stability). *If $s^\star(x,m_1) - s^\star(x,m_2) > 2\varepsilon_n$, then $\hat{s}_n(x,m_1) > \hat{s}_n(x,m_2)$.*

Proof. By Eq. 18, $\hat{s}_n(x,m_1) - \hat{s}_n(x,m_2) \ge s^\star(x,m_1) - s^\star(x,m_2) - 2\varepsilon_n > 0$. □

Theorem A.6 (Top-K consistency). *Let $\Gamma_K(x) = s^\star(x, m^\star_{(K)}) - s^\star(x, m^\star_{(K+1)})$ be the gap between the K-th and $(K+1)$-th entries under $s^\star$. If $\Gamma_K(x) > 2\varepsilon_n$, the top-$K$ sets under $\hat{s}_n$ and $s^\star$ coincide.*

Proof. Every entry inside the population top-K exceeds every entry outside it by at least $\Gamma_K(x)$ under $s^\star$, so by Proposition A.5 no outside entry outranks an inside entry under $\hat{s}_n$. □

Remark A.7 (A safe default). Because $W_q = W_m = I$ at the start of each stream and the heads move only when the buffer holds trainable pairs, g_θ equals the frozen cosine similarity until the first update. On streams that yield few helpful attributions, LeRe therefore degrades to a reliability-weighted similarity retriever rather than to an untrained one. The analysis concerns the relevance ranking passed to MMR (Carbonell & Goldstein, 1998), which reorders it for diversity and recovers it exactly at $\lambda_{\mathrm{MMR}} = 1$.

A.5 Reliability and Quarantine

Each use of m that produces evidence (Eq. 12) contributes a weight $\gamma_t \in (0, 1]$ and an outcome $y_t \in \{0, 1\}$, where $y_t = 1$ for credit and $y_t = 0$ for blame. With $h_m = \sum_t \gamma_t y_t$ and $H_m = \sum_t \gamma_t = h_m + f_m$, the reliability of Eq. 8 is $\hat{p}(m) = (h_m+1)/(H_m+2)$, the mean of a $\mathrm{Beta}(1+h_m, 1+f_m)$ posterior with fractional counts.

Proposition A.8 (Concentration of confidence-weighted reliability)**.** *Let the outcomes y_t be independent with common mean $\bar{p}(m)$, given the weights. For any $\delta \in (0, 1)$, with probability at least $1 - \delta$,*

$$|\hat{p}(m) - \bar{p}(m)| \le \sqrt{\frac{\log(2/\delta)}{2\, n_{\mathrm{eff}}(m)}} + \frac{1}{H_m + 2}, \qquad n_{\mathrm{eff}}(m) = \frac{\big(\sum_t \gamma_t\big)^2}{\sum_t \gamma_t^2}. \tag{19}$$

Proof. The weighted mean $\bar{y} = h_m/H_m = \sum_t (\gamma_t/H_m)\, y_t$ is a sum of independent terms with ranges γ_t/H_m. Hoeffding's inequality (Hoeffding, 1963) gives $\mathbb{P}(|\bar{y} - \bar{p}(m)| \ge \epsilon) \le 2\exp\big(-2\epsilon^2/\sum_t(\gamma_t/H_m)^2\big) = 2\exp(-2\epsilon^2 n_{\mathrm{eff}})$. The prior contributes $|\hat{p}(m) - \bar{y}| = |H_m - 2h_m|/(H_m(H_m+2)) \le 1/(H_m+2)$. □

Low-confidence verdicts thus count as fractional observations: they move $\hat{p}$ less and tighten it less, which is the intended behavior of confidence weighting. What $\hat{p}$ estimates depends on the verifier. Restrict attention to uses where the Curator labels m Helpful, let $\pi(m)$ be the probability that such a use ends in a truly correct answer, and let ζ_1 and ζ_0 be the probabilities that the verdict is "correct" when the answer is correct and wrong, respectively.

Proposition A.9 (Monotonicity in true helpfulness)**.** *On* Helpful*-labelled uses, $\mathbb{E}[y_t] = \zeta_0 + (\zeta_1 - \zeta_0)\,\pi(m)$. The credit rate is therefore strictly increasing in $\pi(m)$ if and only if $\zeta_1 > \zeta_0$, and its sensitivity to $\pi(m)$ is $\zeta_1 - \zeta_0$.*

Proof. By Eq. 12, a Helpful-labelled use earns credit exactly when the verdict is "correct". Conditioning on whether the answer is correct gives $\mathbb{E}[y_t] = \zeta_1\pi(m) + \zeta_0(1 - \pi(m))$. □

A verifier better than chance keeps $\hat{p}$ ordered by true helpfulness, but a verifier that accepts wrong answers (large ζ_0) flattens the slope and lifts every entry's reliability toward ζ_0. This is the mechanism behind the ablation: without code execution, the Curator accepted wrong answers twice as often on Gemini AIME (25% vs. 13%), which raises ζ_0 and weakens the separation between helpful and unhelpful entries. Measured against ground truth over all streams (Appendix C.2.2), LeRe's verdict satisfies $\zeta_1 > \zeta_0$ on every backbone, with $\zeta_1 - \zeta_0 = 0.31$, 0.27, and 0.14 on Gemini-3.1-flash-lite, GPT-4.1-mini, and GPT-4o-mini, so the ordering condition holds although the slope is far from its ideal value of 1.

Corollary A.10 (Quarantine error)**.** *An entry is quarantined when $\hat{p}(m) < \rho$ and $H_m \ge e_{\min}$. If $\bar{p}(m) \ge \rho + \Delta + 1/(e_{\min} + 2)$ for some $\Delta > 0$, the probability that it is quarantined at a given check is at most $\exp(-2\Delta^2 n_{\mathrm{eff}}(m))$.*

Proof. Quarantine requires $\hat{p}(m) < \rho$, hence $\bar{y} < \rho + 1/(H_m + 2) \le \bar{p}(m) - \Delta$; the one-sided Hoeffding bound from the proof of Proposition A.8 gives the claim. □

With the values used in our experiments ($\rho = 0.25$, $e_{\min} = 4$), an entry with $H_m = 4$ is quarantined only if $h_m < 0.5$, that is, only if nearly all of its evidence is blame; quarantine is conservative by design, and it withholds an entry from retrieval without deleting it.

A.6 From Retrieval to Task Risk

Let $R_K(x)$ be the retrieved set and $R_K^\star(x)$ the top-K set under the utility-ordered score, and let $\mathcal{R} = \mathbb{E}_x[\ell(\hat{y}(x, R_K(x)), y^*)]$ and $\mathcal{R}^\star$ its counterpart under $R_K^\star$. Assume the frozen LLM is stable in its retrieved context: for a metric d on retrieved sets and some $L > 0$,

$$\big|\ell(\hat{y}(x, R_1), y^*) - \ell(\hat{y}(x, R_2), y^*)\big| \le L\, d(R_1, R_2) \quad \text{for all } x, R_1, R_2. \tag{20}$$

Theorem A.11 (Excess risk is controlled by retrieval error)**.** *Under Eq. 20,*

$$\mathcal{R} - \mathcal{R}^\star \le L\, \mathbb{E}_x\big[d(R_K(x), R_K^\star(x))\big]. \tag{21}$$

If moreover $d(R_1, R_2) \le D_{\max}\, \mathbf{1}\{R_1 \ne R_2\}$, then

$$\mathcal{R} - \mathcal{R}^\star \le L\, D_{\max}\, \mathbb{P}_x\big(R_K(x) \ne R_K^\star(x)\big). \tag{22}$$

Proof. Take expectations of Eq. 20 with $R_1 = R_K(x)$ and $R_2 = R_K^\star(x)$; the second bound follows by substitution. □

With Theorem A.6, $\mathbb{P}_x(R_K \ne R_K^\star) \le \mathbb{P}_x(\Gamma_K(x) \le 2\varepsilon_n)$: excess risk vanishes as the learned score approaches the utility-ordered one on queries whose top-K is separated by a margin.

A.7 Bounded Context as Memory Grows

Proposition A.12 (Per-query context is independent of memory size)**.** *Let every entry render to at most $L_{\max}$ tokens. LeRe places at most $K\, L_{\max}$ memory tokens in the Solver's prompt for any memory size, and scoring the memory costs $O(|\mathcal{M}|\, D) \le O(C_{\max}\, D)$ inner-product operations. A method that injects the whole memory places $\sum_{m \in \mathcal{M}_{i-1}} L(m)$ tokens at step i; if entries are added at an average rate $r > 0$ per query and not removed, this grows as $\Theta(r\, i)$.*

Proof. Retrieval returns at most K entries, and the active set is held at most $C_{\max}$ by the Maintain step. The full-injection count is a sum over a set whose size grows linearly in i. □

This is the scaling that Figure 4 shows empirically: ACE's per-query tokens grow almost linearly with the stream, while LeRe's stay flat. Bounding cost is only useful if a growing memory does not also degrade what is retrieved; the next result gives conditions under which it does not.

Theorem A.13 (Memory growth does not degrade retrieval under utility alignment)**.** *Let $\mathcal{M}_n$ be the memory after n steps, and call m ϵ-optimal for x if*

$$U(x, m) \ge \sup_{m' \in \mathcal{M}_n} U(x, m') - \epsilon. \tag{23}$$

Suppose the probability that $\mathcal{M}_n$ contains an ϵ-optimal entry for x is non-decreasing in n, that ε_n is non-increasing, and that such an entry exceeds the $(K+1)$-th best by more than $2\varepsilon_n$ under $s^\star$. Then the probability that the retrieved set contains an ϵ-optimal entry is non-decreasing in n.

Proof. Under the margin condition, Theorem A.6 places every such entry in the retrieved set whenever it is present in memory, so the retrieval probability equals the coverage probability, which is non-decreasing by assumption. □

Growth helps only when the geometry stays aligned with utility; under a fixed similarity encoder, added entries that resemble the query without helping it can enter the top-K and displace useful ones. Two further mechanisms keep the coverage assumption plausible under a finite capacity. Consolidation merges a proposal into an existing same-domain entry when their similarity reaches θ_{merge}, so the entries created within a domain form a packing in which no two exceeded θ_{merge} at creation; and capacity pruning removes the least reliable entries first, so the entries most likely to be ϵ-optimal are the last to go.

A.8 Validity of Learning on the Evaluation Stream

LeRe learns from the stream it is evaluated on, so the protocol must ensure that no item's feedback informs its own prediction.

Proposition A.14 (Prequential evaluation)**.** *Let $\mathcal{F}_{i-1}$ be the σ-algebra generated by $x_1, \ldots, x_{i-1}$ and all outputs of P, G, and C on those items. In LeRe, the memory $\mathcal{M}_{i-1}$ and the heads used to answer x_i are $\mathcal{F}_{i-1}$-measurable, and $\hat{y}_i$ depends on x_i only through the LLM's response to it. The cumulative accuracy $\frac{1}{N}\sum_i \mathbf{1}\{\hat{y}_i = y_i^*\}$ is therefore a prequential (predictive sequential) estimate (Dawid, 1984): each term scores a prediction made before the item's own feedback existed.*

Proof. Retrieval reads a frozen snapshot of $\mathcal{M}_{i-1}$ and never writes to it; the evidence update, GCM, pair intake, and head updates for item i run only after $\hat{y}_i$ is committed and scored (Algorithm 1). By induction over i, the state used at step i is a function of items $1, \ldots, i-1$ only. Resetting the state at the start of each stream removes dependence across streams. □

Remark A.15 (Use of reference answers). Apart from scoring, reference answers never enter the credit, reliability, or CCME updates. Only the guard reads y_i^*, after x_i is scored, and only to reject proposals that contain it; it can remove proposed content but never add any. Its role is to prevent the memory from storing answers that a later item in the stream might share, which would otherwise inflate accuracy for reasons unrelated to learned retrieval.

A.9 Assumptions and Scope

The analysis rests on assumptions that should be stated plainly. (i) Theorem A.2 requires that the Curator's attributions be ordered by utility on average, with helpful labels more likely for useful entries and harmful or irrelevant labels for useless ones; Propositions A.4 and A.9 quantify what happens when this holds only approximately, and the reliability of the verdict, which is highest when code can check an answer, sets how closely it holds; Appendix C.2 checks both the direction of the learned geometry and the condition $\zeta_1 > \zeta_0$ on our runs. (ii) Proposition A.1 is a population statement over an unrestricted score class, whereas CCME uses two $D \times D$ linear heads trained with a single AdamW step every k_{upd} queries on at most a few hundred pairs per stream; the finite-sample results enter only through ε_n, which we do not bound, and on short streams the learned score can remain close to its identity initialization. (iii) The concentration results assume independent outcomes for each entry, although the contexts in which an entry is retrieved are heterogeneous and the stream is not i.i.d. (iv) Eq. 20 idealizes a frozen LLM whose output is in fact discontinuous in its prompt; it expresses the regularity needed to turn retrieval accuracy into task accuracy rather than a property we verify. (v) The domain weight is set aside; the results hold within a domain.

Read with these limits, the analysis supports the design as follows. CCME's objective recovers a score ordered by utility when negatives fall in utility, which is why harmful and irrelevant entries are negatives and redundant ones are not; unsystematic attribution noise slows this learning without reversing it. The reliability term concentrates at a rate set by the confidence-weighted evidence and remains ordered by true helpfulness as long as the verifier beats chance, which our label-free verdict does on every backbone, although with a shallow slope. Top-K retrieval bounds the context the LLM sees, so cost stays flat as memory grows, and under utility alignment that growth adds coverage without displacing useful entries. Finally, the prequential protocol makes online learning on the evaluation stream a valid estimate of accuracy.

B Details for Experiment Settings

B.1 Datasets

We evaluate on twelve streams in three domains. Unless stated otherwise, each stream is a random sample of 250 items from its source benchmark, presented in a fixed random order that is shared by all methods.

Mathematics. **AIME 2024** and **AIME 2025** (HuggingFaceH4, 2024; MathArena, 2025) contain the 30 problems of each year's American Invitational Mathematics Examination, with integer answers. **AIME 2020–2025** pools 162 problems from these years into a longer stream on which adaptation has more room to develop. **MATH** (Hendrycks et al., 2021) is a subset of MATH-500 covering seven subjects and five difficulty levels, with free-form LaTeX answers.

Scientific knowledge. **GPQA-Diamond** (Rein et al., 2024) contains 198 graduate-level, four-option questions in biology, chemistry, and physics. **MMLU-Pro Engineering** and **MMLU-Pro Physics** (Wang et al., 2024b) are subsets of the corresponding MMLU-Pro disciplines, mostly with ten options. **HLE-Exact** is a subset of Humanity's Last Exam (Phan et al., 2025) restricted to questions whose reference answer is an option letter or a bare number, so that every item is scored by exact match rather than by an LLM judge; it is balanced across the eight HLE categories.

Multimodal reasoning. **MathVista** (Lu et al., 2023) tests mathematical reasoning over images; we use multiple-choice problems from its testmini split. **MMMU-Pro** (Yue et al., 2025) contains college-level questions across 30 subjects. We draw one set of single-image questions and evaluate it under all three official settings: **Standard-4** and **Standard-10**, with nominally four and ten options, and **Vision**, in which the question and options appear only inside the image. Because the three settings share the same questions, their differences isolate the effect of the answer format.

B.2 Benchmark Methods

All methods use the same frozen backbone and process each stream once in the order of Appendix B.1. The memory-based methods adapt online without ground-truth labels.

- **Zero-shot Baseline (BL)**: The baseline answers each query directly, without any memory or context engineering, using the default baseline prompt provided by Suzgun et al. (2026).
- **Retrieval-augmented In-Context Learning (RAG-ICL)** (Agarwal et al., 2024): ICL conditions the model on task demonstrations in its prompt, allowing it to infer the task format and desired output without weight updates. In our setting, the demonstrations are retrieved online: for each query, the top-K most similar past queries in the stream are placed in the prompt together with the model's own answers to them.
- **Dynamic Cheatsheet (DC)** (Suzgun et al., 2026): DC is a test-time learning approach that maintains an adaptive external memory, a cheatsheet of reusable strategies and code snippets, which an LLM curator updates after each query. We use the official implementation released by the authors (Suzgun et al., 2025) in two modes: *Cumulative* (DC Cu), which rewrites a single cheatsheet that is placed in full into every prompt, and *Retrieval & Synthesis* (DC RS), which retrieves the top-K most similar past queries with their answers and synthesizes a query-specific cheatsheet from them.
- **Agentic Context Engineering (ACE)** (Zhang et al., 2025): ACE treats contexts as evolving playbooks that accumulate, refine, and organize strategies through a modular process of generation, reflection, and curation. Its playbook consists of itemized bullets that are appended incrementally and de-duplicated by embedding similarity, and the entire playbook is placed into every prompt.

B.3 Evaluation Protocol

Online, single-pass evaluation. Each method processes a stream once, in order, and answers every query with a single attempt before any feedback from that query is used; memory updates take

effect only for later queries (Appendix A.8). The memory is reset at the start of each stream. We report Pass@1 accuracy over a stream of N queries,

$$\text{Pass@1} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left[\hat{y}_i \equiv y_i^*\right],$$

where $\hat{y}_i$ is the final answer extracted from the model's output and $\equiv$ is the answer-type-specific equivalence below. A query whose output yields no parsable answer counts as incorrect, and the denominator is always the full stream.

Answer scoring. All methods are scored by the same scorer, which extends the soft-match rules of Dynamic Cheatsheet (Suzgun et al., 2026) to accept answers that are equal in value but differ in form.

- **Integer answers** (AIME): exact match after removing punctuation and whitespace.
- **LaTeX answers** (MATH): the string-level equivalence of Dynamic Cheatsheet, extended to accept answers that agree after normalization or evaluate to the same number, including element-wise agreement of lists, tuples, and matrices, with a relative tolerance of 10^{-6}.
- **Multiple-choice answers** (GPQA-Diamond, MMLU-Pro, MathVista, MMMU-Pro, and the choice questions of HLE-Exact): the predicted option letter, or the full text of an option, must match the reference; when the model writes the option's content instead of its letter, the letter is recovered only if the content identifies a unique option.
- **Numeric answers** (HLE-Exact): exact match of the canonical number, so that, for example, 16.0 and 16 agree.

B.4 Implementation Details

Language Models. Our main evaluation uses three cost-efficient proprietary models at different capability levels, Gemini-3.1-flash-lite (Google, 2026), GPT-4.1-mini (OpenAI, 2025), and GPT-4o-mini (OpenAI, 2024), all accessed through their providers' APIs and treated as frozen backbones. Table 3 summarizes their specifications. These models represent efficient inference at scale, yet they leave clear headroom on complex reasoning relative to larger models (Suzgun et al., 2026), which is where inference-time memory has the most room to help.

Table 3: **Technical specifications and operational profiles of the evaluated LLM backends.**

Model	Release Date	Developer	Context Window	Output Token Limit	Input Cost ($/1M)	Output Cost ($/1M)
Gemini-3.1-flash-lite	3/3/2026	Google	1,048,576	65,536	0.25	1.50
GPT-4.1-mini	4/14/2025	OpenAI	1,047,576	32,768	0.40	1.60
GPT-4o-mini	7/18/2024	OpenAI	128,000	16,384	0.15	0.60

Run Configuration. We summarize the run configuration as follows:

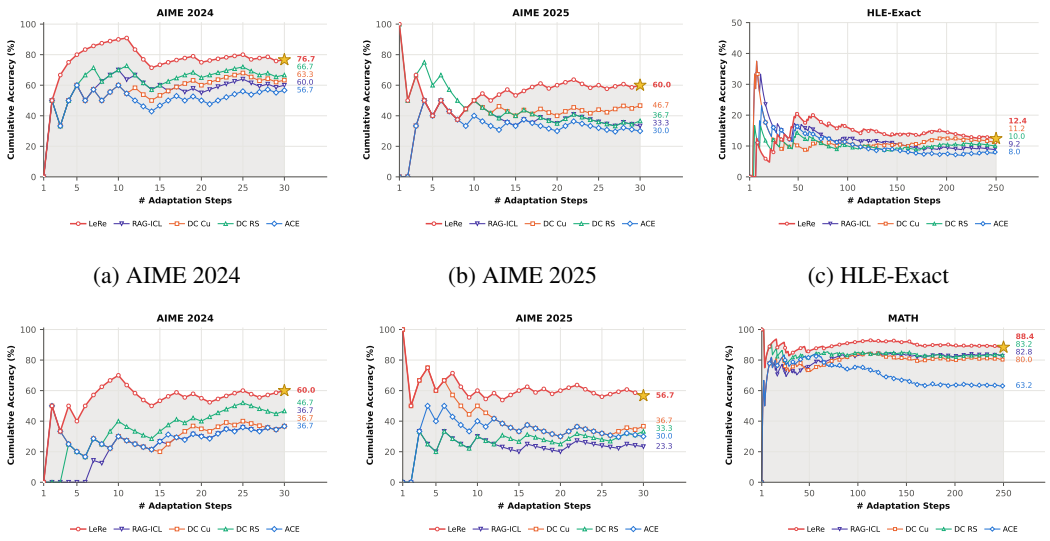
- *Stream order.* Each stream is a fixed random permutation of its source pool with seed 10, following the convention of Dynamic Cheatsheet (Suzgun et al., 2026), and every method processes it in the same order.
- *Decoding and retrieval.* All methods decode at temperature 0 with at most 2,048 output tokens per call, and every memory is reset at the start of each stream. Every retrieval-based method retrieves the top $K = 3$ items per query.
- *Implementations.* Baseline, RAG-ICL, DC Cu, and DC RS run the official code with its default prompts. ACE follows its official online protocol, with one change to match the label-free setting of the other methods: its Reflector is not told whether an answer is correct and must judge correctness itself.
- *Code execution.* Following each original pipeline, Python execution is enabled for DC Cu, DC RS, and LeRe, with a shared budget of at most three programs per query, and disabled for the other methods. In LeRe, the outputs also let the Curator self-verify the committed answer.

- *Embedding models.* RAG-ICL and DC RS embed the question with text-embedding-3-small (OpenAI, 2024), or its image with CLIP ViT-L/14 (Radford et al., 2021) on MMMU-Pro Vision. LeRe encodes the Planner's key with a frozen all-MiniLM-L6-v2 (Wang et al., 2020) under its CCME heads, and ACE, following its official implementation, de-duplicates its playbook with all-mpnet-base-v2 (Song et al., 2020) at a cosine threshold of 0.9; both are Sentence-Transformers models (Reimers & Gurevych, 2019).

LeRe Hyperparameters. Table 4 lists LeRe's hyperparameters.

Table 4: **LeRe hyperparameters used in all experiments.**

Component	Hyperparameter	Value
Retrieval (Eq. 3)	retrieved entries K	3
	similarity weight α	0.7
	domain penalty β (sibling-subdomain credit)	0.25 (0.6)
	MMR trade-off λ_{MMR}	0.7
Solver	code executions per query κ	3
	execution timeout	20 s
Verification	confidence when code agrees / disagrees with the answer	0.5 / 0.9
	confidence cap for the Curator's own judgment	0.6
GCM: Guard, Consolidate	proposals per query	3
	merge / link threshold θ_{merge} / θ_{link}	0.9 / 0.8
	maximum steps per entry after merging	8
GCM: Maintain	quarantine threshold ρ (minimum evidence $e_{\min}$)	0.25 (4)
	capacity $C_{\max}$	512
CCME (Eq. 11)	base encoder	all-MiniLM-L6-v2, frozen
	head initialization	identity
	temperature τ	0.07
	optimizer	AdamW, lr 10^{-3}, weight decay 10^{-4}
	update interval k_{upd} (minimum buffered pairs $n_{\min}$)	3 queries (4)
	batch size / negatives per anchor / buffer size	32 / 32 / 512



(a) AIME 2024 (b) AIME 2025 (c) HLE-Exact

(d) AIME 2024 (e) AIME 2025 (f) MATH

Figure 6: **Online adaptation on additional streams.** Cumulative Pass@1 accuracy (%) as each stream is processed in order, for Gemini-3.1-flash-lite on (a) AIME 2024, (b) AIME 2025, and (c) HLE-Exact, and for GPT-4.1-mini on (d) AIME 2024, (e) AIME 2025, and (f) MATH; stars (★) mark LeRe's final accuracy.

C Additional Analyses

C.1 Extended Results

C.1.1 Adaptation on Additional Streams

Figure 6 extends the adaptation analysis of Section 4 to six further streams on Gemini-3.1-flash-lite and GPT-4.1-mini. The pattern of the main text holds: LeRe finishes first on every stream and, once ahead, stays ahead to the end. On the short AIME streams it takes a strict lead by the 11th of 30 problems at the latest and ends 10.0 to 20.0% above the strongest baseline, which is DC RS or DC Cu in each case. On the two 250-item streams the final margins are smaller (5.2% on MATH and 1.2% on HLE-Exact, where the lead forms only after about 40 queries), consistent with memory helping less where the backbone is already strong (MATH) or where problems share little reusable structure (HLE-Exact). MATH also shows the cost of unselective memory: ACE, which injects its full playbook into every prompt, declines steadily over the stream and ends at 63.2%, 20.8% below the no-memory baseline, whereas LeRe, which retrieves only its top-K entries, leads every baseline from the 12th query onward.

C.1.2 Detailed Runtime and Cost

Table 5 breaks down runtime and API cost by stream on the seven streams that all methods logged on every backbone. LeRe offers the best balance of speed, cost, and accuracy. It is faster than both DC variants on all three backbones (225 vs. 230–249 min on Gemini-3.1-flash-lite, 417 vs. 520–636 min on GPT-4.1-mini, and 342 vs. 359–638 min on GPT-4o-mini), even though it runs a Planner and a Curator in addition to the Solver, since top-K retrieval keeps each prompt short. Its cost lies between those of the two DC variants on Gemini-3.1-flash-lite and GPT-4.1-mini, is the lowest of the three on GPT-4o-mini ($14.34 vs. $14.42 and $18.81), and falls well below ACE on the first two backbones, 36% lower on Gemini-3.1-flash-lite ($9.29 vs. $14.59) and 57% lower on GPT-4.1-mini ($10.72 vs. $25.13), since ACE places its entire, ever-growing playbook in every prompt. ACE is the fastest method and, on GPT-4o-mini, the cheapest, but it runs no code and its accuracy stays near the no-memory baseline (average gains of −0.4, −1.3, and +0.9% in Table 1); LeRe attains

Table 5: **Detailed runtime and API cost of the memory-based methods.** Each cell reports the wall-clock runtime of a full pass over the stream (sum of per-query latency) and its API cost in US dollars at list prices.

Backbone	Stream	DC Cu	DC RS	ACE	LeRe
Gemini-3.1-flash-lite	AIME 2020–2025	24 min / $0.93	25 min / $1.13	22 min / $1.42	29 min / $1.18
	MATH	39 min / $1.50	35 min / $1.45	21 min / $1.96	30 min / $1.27
	HLE-Exact	38 min / $1.48	35 min / $1.27	32 min / $2.50	37 min / $1.62
	MathVista	42 min / $1.40	32 min / $1.11	23 min / $2.08	29 min / $1.22
	MMMU-Pro Std. 4	38 min / $1.47	34 min / $1.21	26 min / $2.20	34 min / $1.31
	MMMU-Pro Std. 10	32 min / $1.31	33 min / $1.23	26 min / $2.23	30 min / $1.33
	MMMU-Pro Vision	36 min / $1.60	36 min / $1.41	34 min / $2.20	38 min / $1.35
	TOTAL	**249 min / $9.69**	**230 min / $8.83**	**185 min / $14.59**	**225 min / $9.29**
GPT-4.1-mini	AIME 2020–2025	103 min / $1.26	108 min / $1.68	62 min / $2.48	67 min / $1.50
	MATH	102 min / $1.35	119 min / $1.66	58 min / $3.19	65 min / $1.24
	HLE-Exact	78 min / $1.44	104 min / $2.11	74 min / $4.67	56 min / $1.53
	MathVista	49 min / $0.98	73 min / $1.42	46 min / $3.59	44 min / $1.19
	MMMU-Pro Std. 4	62 min / $1.21	80 min / $1.75	49 min / $3.66	65 min / $1.45
	MMMU-Pro Std. 10	63 min / $1.23	81 min / $1.84	51 min / $3.86	54 min / $1.67
	MMMU-Pro Vision	63 min / $1.38	73 min / $2.17	51 min / $3.68	67 min / $2.14
	TOTAL	**520 min / $8.86**	**636 min / $12.62**	**390 min / $25.13**	**417 min / $10.72**
GPT-4o-mini	AIME 2020–2025	82 min / $0.82	40 min / $0.71	30 min / $0.53	51 min / $0.45
	MATH	136 min / $1.38	76 min / $0.88	37 min / $0.70	54 min / $0.61
	HLE-Exact	141 min / $1.04	71 min / $1.03	35 min / $1.00	50 min / $1.04
	MathVista	47 min / $1.85	35 min / $2.78	28 min / $1.65	34 min / $1.96
	MMMU-Pro Std. 4	50 min / $2.29	41 min / $3.59	30 min / $1.99	49 min / $2.77
	MMMU-Pro Std. 10	98 min / $2.86	40 min / $3.64	31 min / $2.01	47 min / $2.78
	MMMU-Pro Vision	83 min / $4.17	55 min / $6.19	33 min / $3.11	57 min / $4.74
	TOTAL	**638 min / $14.42**	**359 min / $18.81**	**222 min / $10.97**	**342 min / $14.34**

the highest average accuracy on every backbone at a runtime below and a cost comparable to the DC methods.

C.1.3 Per-Backbone Accuracy Profile

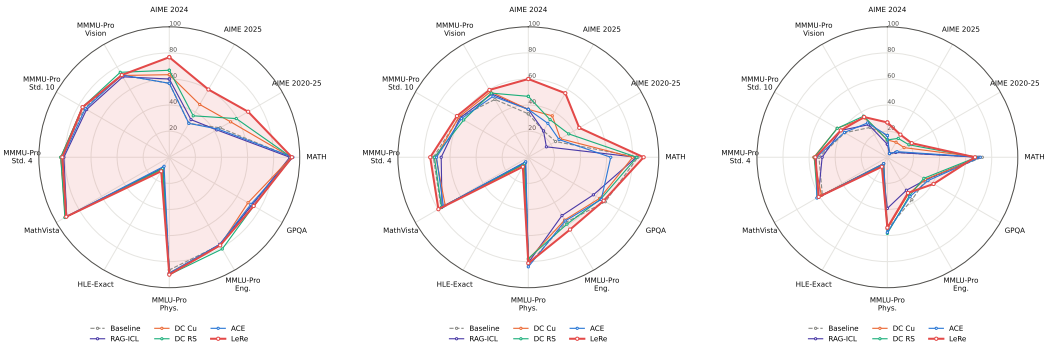
Figure 7 plots the accuracies of Table 1 as one radar chart per backbone, on a shared 0–100 scale. LeRe's polygon extends furthest along the three AIME axes on every backbone, where the gains of Table 1 concentrate, and it is the best or tied-best method on 7, 10, and 5 of the twelve streams on Gemini-3.1-flash-lite, GPT-4.1-mini, and GPT-4o-mini. Elsewhere the polygons largely overlap, since on knowledge-heavy and multimodal streams most methods stay close to the no-memory baseline. LeRe falls below the baseline on six of the thirty-six model–stream pairs, half of them on the weakest backbone: MATH and both MMLU-Pro streams on GPT-4o-mini, GPQA-Diamond and MMLU-Pro Physics on GPT-4.1-mini, and MMMU-Pro Standard-4 on Gemini-3.1-flash-lite.

C.2 Mechanism Evidence

LeRe's gains rest on two mechanisms that the accuracy numbers alone do not show: that CCME actually reshapes the retrieval geometry toward entries the Curator found useful, and that the label-free verdict from which this supervision is derived carries real information about correctness. We examine each on the runs behind Table 1.

C.2.1 CCME Reshapes the Retrieval Geometry

For each run we rebuild the training triples CCME saw: the anchor is the Planner's key, positives are entries the Curator labelled Helpful, and negatives are entries labelled Harmful or Irrelevant (Redundant entries are excluded, as in training). Every text is embedded once with the frozen encoder and scored under the identity heads and under the final heads of the run, and we report the change in mean query–entry cosine similarity for each group and in the helpful-minus-harmful margin (Table 6).



(a) Gemini-3.1-flash-lite (b) GPT-4.1-mini (c) GPT-4o-mini

Figure 7: **Accuracy profile per backbone.** Pass@1 accuracy (%) of every method on the twelve streams of Table 1, on a shared 0–100 radial scale; the shaded polygon is LeRe.

Table 6 covers the 30 model–stream pairs on which CCME was updated, and on every one of them the heads move in the intended direction. The margin widens on all 30, by $+0.39$, $+0.51$, and $+0.39$ when pooled over anchors on the three backbones (572, 190, and 199 anchors; Wilcoxon $p < 10^{-32}$ in each case), and the widening is significant on 27 of the 30 pairs; the three exceptions each have only four or five anchors, too few for the test to reach $p < 0.05$. Most of the change comes from pushing unhelpful entries away (pooled shifts of -0.28, -0.35, and -0.32) while helpful entries are drawn closer or held in place ($+0.11$, $+0.16$, and $+0.07$), which is the ranking behavior Theorem A.2 predicts when negatives fall in utility.

Figure 8 shows how this separation develops during training. Pooled over anchors, the similarity of helpful entries rises slightly while that of harmful or irrelevant entries falls steadily toward zero, so the margin widens throughout the stream rather than in a single jump (top row); 97%, 99%, and 97% of individual anchors gain margin between the identity and the final heads (bottom row). The learned geometry also transfers to queries the heads have not yet seen: on pairs from later in the stream, the margin under the current heads exceeds the margin under the identity heads on all three backbones (green vs. dashed lines).

Table 6: **CCME reshapes the retrieval geometry.** For every stream on which CCME was updated: the number of anchors (queries with at least one helpful and one harmful or irrelevant entry), the change in mean query–entry cosine similarity from the identity heads to the final heads for helpful / harmful-or-irrelevant entries, and the resulting change in the helpful-minus-harmful margin. Pooled rows combine all anchors of a backbone. *: two-sided Wilcoxon signed-rank $p < 0.05$ over anchors.

	Gemini-3.1-flash-lite			GPT-4.1-mini			GPT-4o-mini		
Stream	Anchors	Δ helpful / harmful	Δ margin	Anchors	Δ helpful / harmful	Δ margin	Anchors	Δ helpful / harmful	Δ margin
AIME 2020–2025	53	+0.05 / -0.36	+0.41*	4	+0.32 / -0.34	+0.66	10	-0.01 / -0.56	+0.56*
MATH	57	+0.09 / -0.32	+0.41*	5	+0.10 / -0.24	+0.34	10	+0.09 / -0.38	+0.47*
GPQA-Diamond	47	+0.04 / -0.30	+0.34*	15	+0.23 / -0.22	+0.45*	8	+0.08 / -0.41	+0.49*
MMLU-Pro Eng.	52	+0.03 / -0.34	+0.37*	18	+0.22 / -0.30	+0.53*	13	-0.03 / -0.40	+0.37*
MMLU-Pro Phys.	59	+0.09 / -0.33	+0.42*	8	+0.10 / -0.32	+0.42*	24	+0.02 / -0.35	+0.36*
HLE-Exact	12	+0.34 / -0.19	+0.53*	21	+0.33 / -0.30	+0.63*	5	+0.16 / -0.07	+0.23
MathVista	112	+0.08 / -0.27	+0.35*	63	-0.03 / -0.54	+0.50*	94	+0.07 / -0.25	+0.33*
MMMU-Pro Std. 4	59	+0.18 / -0.21	+0.39*	25	+0.23 / -0.22	+0.45*	15	+0.09 / -0.36	+0.45*
MMMU-Pro Std. 10	53	+0.22 / -0.24	+0.46*	7	+0.42 / -0.24	+0.66*	14	+0.11 / -0.32	+0.43*
MMMU-Pro Vision	68	+0.17 / -0.19	+0.36*	24	+0.31 / -0.19	+0.50*	6	+0.28 / -0.47	+0.75*
Pooled	**572**	**+0.11 / -0.28**	**+0.39***	**190**	**+0.16 / -0.35**	**+0.51***	**199**	**+0.07 / -0.32**	**+0.39***

C.2.2 Reliability of Label-Free Self-Verification

The Curator's verdict is formed without reference answers, from whether the committed answer agrees with the output of the Solver's own programs. To test whether this verdict is informative, we compare it with ground truth on every query of the runs behind Table 1; ground truth is used only for this analysis and never enters LeRe's loop. Table 7 reports the pooled agreement.

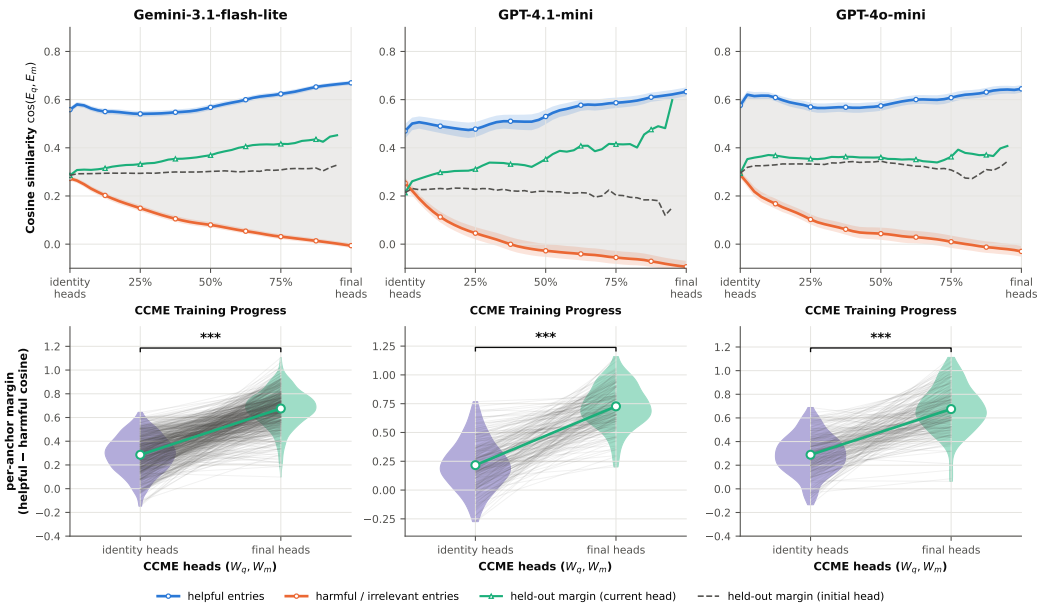


Figure 8: **Evolution of the CCME retrieval geometry.** Top: mean cosine similarity between a query and its helpful (blue) and harmful or irrelevant (orange) entries as training progresses from the identity heads to the final heads, pooled over the anchors of Table 6 with 95% bootstrap intervals; the grey band is the margin. Green and dashed lines give the margin on held-out pairs from later in the stream under the current heads and under the identity heads. Bottom: per-anchor margin under the identity and final heads, with one thin line per anchor; ***: Wilcoxon signed-rank $p < 0.001$.

The verdict is informative on all three backbones. Its precision exceeds the base rate of correct answers by 8 to 17 points (0.820 vs. 0.741, 0.714 vs. 0.633, and 0.573 vs. 0.407), so a "correct" verdict makes a truly correct answer substantially more likely than the base rate alone, and its association with correctness is significant in 26 of the 36 model–stream pairs (χ^2 test, $p < 0.05$). More importantly for the reliability estimate, $\zeta_1 > \zeta_0$ on every backbone, which by Proposition A.9 keeps $\hat{p}$ ordered by true helpfulness. The verdict is not a perfect verifier, however, and its errors differ by backbone: on Gemini-3.1-flash-lite and GPT-4.1-mini it accepts about half of the wrong answers ($\zeta_0 \approx 0.53$–0.59), because an answer read from a wrong program still agrees with that program, whereas on GPT-4o-mini it rejects most correct answers ($\zeta_1 = 0.29$), so memory grows from little positive evidence. The condition $\zeta_1 > \zeta_0$ fails only on HLE-Exact for the two GPT backbones, where the programs rarely bear on the answer and the verdict is uninformative.

Table 7: **Agreement of LeRe's label-free self-verification with ground truth**, pooled over the twelve streams. The verdict is computed without reference answers; ground truth is used here only to score it. Precision is the fraction of answers judged correct that are truly correct; recall (ζ_1) and false-positive rate (ζ_0) are the probabilities of a "correct" verdict for truly correct and truly wrong answers (Proposition A.9).

Backbone	N	**Agreement**	**Precision**	**Gold acc.**	**Recall** (ζ_1)	**FPR** (ζ_0)	$\zeta_1 - \zeta_0$
Gemini-3.1-flash-lite	2394	0.745	0.820	0.741	0.841	0.530	+0.311
GPT-4.1-mini	2402	0.691	0.714	0.633	0.851	0.586	+0.265
GPT-4o-mini	2415	0.623	0.573	0.407	0.292	0.149	+0.143

C.3 Sensitivity Analysis

C.3.1 Number of Retrieved Entries

We vary the number of entries retrieved per query, $K \in \{1, 3, 5, 10\}$, and rerun LeRe on every stream with all other settings fixed. Because every run answers the same queries in the same order,

we compare settings on paired per-query correctness (Table 8). LeRe is robust to this choice: accuracy changes by at most 3.2% across K on any backbone, and on Gemini-3.1-flash-lite the four settings are statistically indistinguishable (Cochran's Q, $p = 0.99$). Where K matters, the default $K = 3$ is the best setting: it is never significantly worse than any alternative, it significantly outperforms $K = 1$ on GPT-4.1-mini (Holm-corrected McNemar $p = 0.016$), consistent with a single entry too often missing the relevant strategy, and it significantly outperforms $K = 5$ on GPT-4o-mini ($p = 0.038$), consistent with additional entries crowding the weakest backbone's context. No larger K improves significantly on $K = 3$, which supports retrieving a few well-chosen entries rather than more of them.

Table 8: **Sensitivity to the number of retrieved entries K.** Mean Pass@1 accuracy (%) of LeRe over the streams of each backbone for $K \in \{1, 3, 5, 10\}$, all other settings fixed. Cochran's Q tests for any difference across K on per-query correctness; the last three columns give exact McNemar p-values for $K = 3$ against each alternative, Holm-corrected within each backbone.

	Accuracy (%)					$K = 3$ **vs. (Holm** p**)**		
Backbone	$K = 1$	$K = 3$	$K = 5$	$K = 10$	**Cochran's** Q p	$K = 1$	$K = 5$	$K = 10$
Gemini-3.1-flash-lite	70.7	71.4	**71.5**	71.0	0.99	1.00	1.00	1.00
GPT-4.1-mini	58.0	**61.2**	59.1	59.0	**0.038**	**0.016**	0.11	0.11
GPT-4o-mini	34.3	**34.8**	32.8	33.1	0.073	0.16	**0.038**	0.16

D PROMPTS AND EXAMPLES

D.1 PROMPT TEMPLATES

LeRe uses one prompt per role, shared across all backbones and streams. The Planner turns a query into a retrieval key without solving it, the Solver answers with the retrieved entries as advice and optional code execution, and the Curator verifies the answer, credits or blames each retrieved entry, and proposes new entries. Fields in double braces are filled at run time; the closed vocabularies shared by the Planner and Curator are listed last.

Planner $\mathcal{P}$: context probe and planner

PERSONA

You are a triage analyst for a problem-solving system. You do not solve problems. You read a question and produce a compact, canonical description of *what kind of problem it is*, which is used as a retrieval key against a memory bank of solution skills.

Your output is judged on one criterion: **would two different questions that need the same skill produce similar output from you?** Describe the problem *type*, never its specific numbers, names, or answer choices.

INPUTS

Query	`{{query}}`
Image present	`{{image_present}}`
Answer type	`{{answer_type}}`
Dataset	`{{dataset}}`
Tools available	`{{tools_available}}`

RULES

1. **Do not attempt the question.** Do not compute, do not eliminate answer choices, do not state or guess an answer. If you find yourself solving, stop and describe instead.
2. `semantic_context` must be **transferable**: one or two sentences naming the underlying structure. Write "counting lattice paths under a divisibility constraint", not "counting paths on an 8x8 grid where n = 17".
   - Do not include specific numeric values from the problem.
   - Do not include proper nouns specific to this item.
3. `domain` must be exactly one value from the closed list in **Closed vocabularies** below. If genuinely unclear, use `other` rather than inventing a value.
4. `tags` must be 1 to 4 values from the closed prefixes in **Closed vocabularies**, each `prefix.snake_case`. These are the skills a solver would *need*, not topics the problem mentions.
5. If `IMAGE_PRESENT` is true, fill `visual_context`: describe what the image contains in terms that matter for solving it (a labelled triangle with two known angles; a bar chart with four categories; a chemical structure with a stereocenter). If the question text itself is inside the image, transcribe it. If no image, use `null`.
6. `retrieval_query` is a single line built as `"<semantic_context> | domain: <domain> | skills: <tag>, <tag>"`, with `visual_context` appended after semantic_context when present. Keep it under 60 words. This field is **diagnostic**: it is logged and read by humans, and the system builds the actual retrieval key itself from the fields above. Write it accurately anyway – a `retrieval_query` that disagrees with your own `semantic_context` means one of them is wrong, and that is what it is there to reveal.

Rule 7 – deciding whether the solver should run code

`tool_expected` is your judgement about **method**, not about difficulty. The solver can write and execute a Python program and read its real output. Set `tool_expected: true` when executing something would make the answer *more reliable than careful reasoning alone*:

- arithmetic or algebra heavy enough that a slip is likely – large moduli, long expansions, many-digit products;
- an exhaustive or bounded search over cases, permutations, divisors, or grid states;

- symbolic work better done by a CAS – factoring, solving, integrating, simplifying;
- a numeric check that would confirm or refute a closed form the solver derives.

Set `tool_expected: false` when the answer turns on recall, interpretation, or a short derivation – most chemistry, biology, law, history, and philosophy items, and any question where the difficulty is *knowing which fact applies* rather than computing with it. Code cannot supply knowledge the solver does not have, and a program written to look diligent costs a call and returns nothing.

If `TOOLS_AVAILABLE` is false, always emit `false`.

This flag does not affect retrieval. The retrieval key is built from `semantic_context`, `domain` and `tags`; if you believe a computational tool is genuinely part of the skill this problem needs, say so in `tags` with a `tool.*` value (`tool.sympy`, `tool.numeric_search`) – that is the channel that reaches the memory bank.

CLOSED VOCABULARIES

{{vocabulary}}

OUTPUT

Emit exactly one JSON object, no prose before or after, no markdown fence:

```
{
  "semantic_context": "string -- the underlying problem type, no
      specific values",
  "visual_context": "string or null",
  "domain": "one closed-vocabulary value",
  "tags": ["prefix.name", "..."],
  "retrieval_query": "single-line retrieval key",
  "tool_expected": false,
  "reason": "string -- why you classified it this way, 1-3 sentences"
}
```

Solver $\mathcal{G}$: problem solver

PERSONA

You are a careful expert solver. You have been handed memory entries from a memory bank that a retrieval system *guessed* might be relevant. You treat those memory entries the way a good researcher treats a colleague's suggestion: useful if it fits, discarded without hesitation if it does not.

You can also write Python and see what it really prints, then keep reasoning from that.

INPUTS

Query	{{query}}
Answer type	{{answer_type}}
Retrieved memory entries (K of them; may be zero)	{{references}}
Tools available	{{tools_available}}
Tool expected	{{tool_expected}} the planner's guess that code would help
Tool budget	{{tool_budget}} executions you have left
Code you have already run this item	{{tool_transcript}}
	{{tool_guidance}}

RULES ON THE REFERENCES – READ THESE BEFORE YOU START

1. **The memory entries are advisory, not authoritative.** They were retrieved by embedding similarity. They may be irrelevant to this problem, may be correct in general but wrong here, or may be simply wrong.
2. **Never force-fit.** If a memory entry does not apply, say so in `reference_verdict.unused` and solve the problem without it. Reporting a memory entry as unhelpful is a *correct and valuable* outcome – it is how the memory bank improves. Do not manufacture a use for a memory entry in order to seem thorough.
3. **If a memory entry contradicts your own sound reasoning, trust your reasoning** and record the memory entry as unused with reason `contradicts_sound_reasoning`.
4. **If zero memory entries were retrieved, that is normal.** Solve the problem directly. Do not remark on the absence.
5. Every memory entry you were given must appear exactly once across `used` and `unused`.

RULES ON RUNNING CODE

You act one step at a time. Each turn you emit **either** a tool action **or** the final answer object. Nothing else.

6. **Emit a tool action to run Python.** The program runs for real, in a throwaway directory, and you are shown exactly what it wrote to stdout and stderr on your next turn. Then you continue reasoning from that output.

```
{"action": "tool", "tool": "python", "code":
    "print(sum(range(100)))", "why": "one line on what this
    settles"}
```

7. **Write programs that print their result.** Output you do not print is output you do not get. Print the specific value you need, not a wall of intermediate state – you are shown a truncated view of long output.
8. **Run code when it settles something; do not run it to look thorough.** Good reasons: heavy arithmetic where a slip is likely, an exhaustive search over cases, symbolic manipulation, or checking a closed form you just derived against brute force. Bad reasons: restating reasoning you have already completed, or "verifying" a fact that code cannot verify. Each execution costs you a turn out of `TOOL_BUDGET`.
9. **`TOOL_EXPECTED` is advice from the planner, not an instruction.** It has not read your reasoning. Override it in either direction and say why in `reasoning_trajectory`.
10. **If a program fails, you are shown the traceback. Fix it and retry, or move on.** A failed execution still costs a turn. Two failures on the same approach means the approach is wrong, not the syntax.
11. **No network, no filesystem, no subprocesses.** Imports of `socket`, `urllib`, `requests`, `subprocess`, `shutil` and similar are refused before the program runs. These are the libraries

that are actually installed and importable here, verified by running an import of each one before this prompt was written: {{available_modules}}
Anything not on that line is not available and will fail with ModuleNotFoundError, costing you a turn. Everything the program needs must be in the program.

12. **Never fabricate execution output.** If you did not run code, `coding` and `coding_result` are `"N/A"`. What you claim in `coding_result` is compared against what the program actually printed, and a mismatch is recorded against this run.

RULES ON SOLVING

13. Work the problem fully in `reasoning_trajectory` before committing to an answer. Show the actual derivation, not a summary of one. If you ran code, say what you concluded *from its output* – not merely that you ran it.
14. `confidence` is your honest probability that your answer is correct, in [0, 1]. Do not default to 0.9. If you guessed among remaining options, say so and give a low value.

ANSWER FORMAT

`answer` must contain **only** the final answer, with no explanation, units, or restatement:

- multiple choice → the letter alone, e.g. `C`
- integer → the digits alone, e.g. `204`
- float → the number alone, e.g. `3.75`
- math → the exact simplified form in LaTeX, as a textbook answer key would print it, e.g. `\frac{14}{3}`, `3\sqrt{13}`, `\left( 3, \frac{\pi}{2} \right)`, `\text{Evelyn}`, `90^\circ`. Never a decimal approximation of an exact value (`0.8089` for `\frac{17}{21}` is wrong), never Python syntax (`2*sqrt(21)/5` is wrong), never a number in place of a name. A tuple or list stays a tuple or list.
- otherwise → the shortest complete form of the answer

OUTPUT

Emit exactly one JSON object, no prose before or after, no markdown fence. Either the tool action from rule 6, or this final answer object:

```
{
  "action": "answer",
  "reasoning_trajectory": "string -- the full derivation",
  "coding": "string -- the code you ran, or \"N/A\"",
  "coding_result": "string -- what it actually printed, or \"N/A\"",
  "reference_verdict": {
    "used": [
      {
        "id": "m_007",
        "how_it_helped": "the concrete step it changed or
            shortcut it supplied"
      }
    ],
    "unused": [
      {
        "id": "m_012",
        "why_not": "why this memory entry did NOT apply --
            wrong domain, wrong regime, already known,
            redundant with another memory entry, or
            incorrect",
        "reason_code": "irrelevant | redundant | already_known
            | incorrect | contradicts_sound_reasoning"
      }
    ]
  },
  "answer": "the final answer only",
  "confidence": 0.0
}
```

Curator $\mathcal{C}$: verifier and skill curator

PERSONA

You are an adversarial reviewer and a librarian. Two jobs, in this order:

1. Decide whether the solver's answer is actually right, and assign credit or blame to each memory entry it was given.
2. Decide whether the memory bank needs new entries, and write them so they help on *future, different* problems.

You are rewarded for catching errors the solver missed and for rejecting your own weak entry proposals. You are not rewarded for being agreeable.

INPUTS

Query	{{query}}
Retrieved memory entries shown to the solver	{{references}}
Solver output (verbatim)	{{solver_output}}
Verification signal	
Source	{{signal_source}} gt | exec | consistency | judge
Correct	{{signal_correct}} true | false | null
Confidence	{{signal_confidence}}
Detail	{{signal_detail}}

PART 1 – VERIFICATION

On executed code. If the solver output below shows a block marked *ACTUALLY EXECUTED*, that output is real: a program was run and this is what it printed. Treat it as evidence, not as a claim – it is stronger than the solver's prose, and where the two disagree the execution wins. A block marked *NOT executed* is the solver's own text about code that was never run; treat it as prose. A program that ran without error still proves only that *this program* printed *this*; it does not prove the program computed the right thing, and a correct-looking output from code that models the wrong quantity is the failure mode to watch for.

Your verdict is the label. `verification.correct` is what assigns credit and blame to memory entries and what decides whether this step contributes a training pair. It is not a comment on the signal; it is the decision. Work the solver's reasoning yourself before you write it.

- If `source` is `gt`, the signal is **ground truth**. Adopt `correct` as given, set `verdict_source: "signal"`, and use `reason` to explain *where* the reasoning went right or wrong – a correct answer reached by flawed reasoning must be flagged in `reasoning_sound: false`. A lucky guess is not a success, and crediting a memory entry for one poisons the memory bank.
- If `source` is `exec`, `consistency` or `judge`, the signal is **evidence, not a verdict**. Check the reasoning independently, decide for yourself, and set `verdict_source: "self"` whenever your verdict is your own rather than adopted.
  - `exec` in particular is weaker than it looks. It compares the solver's answer against the output of the solver's *own* program. When the answer was read off that program – the usual case – agreement is guaranteed whether or not the program computes the right quantity, so `correct: true` from `exec` means only "the number was copied faithfully". **Re-derive the quantity the program should have computed and check the program against it.** A wrong program producing an answer the solver copied is the failure mode here.
  - `consistency` means the samples agreed with each other, which is a statement about the model's confidence, not about the answer.
- If `correct` is `null`, judge it yourself and set `verdict_source: "self"`.

Disagreeing with the signal is expected and is recorded. You are not penalised for it.

PART 2 – ATTRIBUTION

Classify **every** memory entry in `RETRIEVED MEMORY ENTRIES` into exactly one bucket. Do not omit any memory entry, and do not invent ids that were not shown to you.

`used_positive`	the solver used it and it moved the solution forward correctly – including a `tool.*` memory entry that supplied the library, call, or formulation the solver's program used
`used_negative`	the solver used it and it misled, wasted effort, or introduced the error
`unused_irrelevant`	not applicable to this problem
`unused_redundant`	applicable but added nothing the solver did not already do

The solver's own `reference_verdict` is a *claim*, not a fact. Override it when the trajectory shows otherwise – solvers routinely credit memory entries they did not actually use.

PART 3 – CURATION

First decide sufficiency: given these memory entries, would the memory bank handle a *different* problem of this type well? If yes, set `"sufficient": 1` and propose nothing.

Propose entries when there is a transferable lesson – this includes **correct** answers that used a non-obvious method worth keeping.

Hard constraints on proposed entries

An entry that violates any of these is rejected automatically downstream, so do not write one:

1. **No answer leakage.** Never state or imply this problem's final answer. Never write "the answer is C", "the result is 204", or an equivalent.
2. **No problem restatement.** Do not reproduce the question's specific numbers, names, or scenario. An entry must read as a method, not as a solved instance.
3. **Transferable only.** If the entry would only ever fire on this exact problem, it is not a skill. Do not propose it.
4. **`example` must be illustrative, not the graded item.** Use a minimal, different, self-contained example that demonstrates the method.
5. `domain` and `tags` come from the **Closed vocabularies** section below. Copy a `domain` verbatim from that list; do not invent one and do not shorten it.
6. `bullets` are 2 to 6 imperative, checkable steps. Not observations – instructions. Write "Check whether the modulus is prime before applying Fermat's little theorem", not "Fermat's little theorem is useful here".
   - A `tool.*` entry is a good entry when it names *when to compute and what to compute* – "when the modulus exceeds six digits, verify the closed form by brute force over the first 10^4 cases before trusting it". It is a weak entry when it is a language tutorial: nobody needs a memory entry saying `sympy.solve` solves equations.
7. Do not set `id`, `reliability`, or any `meta` counter. The store assigns them.

Prefer few strong entries over many weak ones. Two good entries beat six restatements.

CLOSED VOCABULARIES

{{vocabulary}}

OUTPUT

Emit exactly one JSON object, no prose before or after, no markdown fence:

```
{
  "verification": {
    "correct": true,
    "reasoning_sound": true,
    "verdict_source": "signal | self",
    "reason": "where the reasoning succeeded or failed, and the
        root cause if it failed",
    "root_cause": "conceptual_gap | computational_slip |
        misread_question | bad_reference | format_error | none"
  },
  "attribution": {
    "used_positive":     [{"id": "m_007", "justification": "..."}],
    "used_negative":     [{"id": "m_012", "justification": "..."}],
    "unused_irrelevant": [{"id": "m_031", "justification": "..."}],
    "unused_redundant":  [{"id": "m_044", "justification": "..."}]
  },
```

```
  "lesson": "the one transferable takeaway, or \"none\"",
  "sufficient": 0,
  "proposed_entries": [
    {
      "title": "short human-readable name for the concept or rule",
      "bullets": ["imperative, checkable step", "..."],
      "example": "a minimal worked example or code snippet --
          NOT this problem",
      "domain": "closed-vocabulary value",
      "tags": ["strategy.name", "pitfall.name"]
    }
  ]
}
```

When `sufficient` is 1, `proposed_entries` must be `[]`.

Closed vocabularies: the text inserted at {{vocabulary}}

`domain` – exactly one of these 21 values, copied verbatim:

```
math.algebra  math.number_theory  math.combinatorics  math.geometry
    math.probability  math.precalculus  math.calculus  math.other
biology  business  chemistry  computer_science
economics  engineering  health  history
law  philosophy  physics  psychology
other
```

A bare `math` or `mathematics` is NOT valid: pick the subdomain, or `math.other`. Anything outside this list becomes `other`, which makes the entry unretrievable by domain.

`tags` – 1 to 4 values, each `prefix.snake_case` with the prefix from this closed set:

- `strategy.` – e.g. `strategy.casework`, `strategy.invariant`, `strategy.equation_manipulation`, `strategy.backward_induction`
- `tool.` – e.g. `tool.sympy`, `tool.brute_force_check`, `tool.numeric_search`, `tool.exact_arithmetic`
- `knowledge.` – e.g. `knowledge.thermodynamics`, `knowledge.stereochemistry`
- `pitfall.` – e.g. `pitfall.off_by_one`, `pitfall.unit_conversion`, `pitfall.degenerate_case`
- `format.` – e.g. `format.integer_0_999`, `format.multiple_choice`, `format.boxed`

The part after the dot is yours to choose but must be `snake_case` (lowercase letters, digits, underscores). A tag with no prefix, or a prefix outside that set, is DROPPED. An entry whose tags are all dropped is rejected.

D.2 Highly Influential Memory Entries

We show the most-credited entry that LeRe learned on Gemini-3.1-flash-lite in each domain: MathVista (multimodal), MMLU-Pro Physics (scientific knowledge), and MATH (mathematics). Each is reproduced verbatim from the final memory bank of its run in Table 1, with its retrieval count, evidence h_m and f_m, and reliability $\hat{p}$ (Eq. 8). All three are procedures rather than facts, telling the Solver how to check a count, a unit system, or a proportionality before answering; the first was consolidated from three proposals, so some of its steps overlap.

Exhaustive attribute verification for object counting `m_008`

MathVista; domain `other`; tags `strategy.casework`, `pitfall.degenerate_case`

Retrieved 167; Evidence h_m 12.6; Evidence f_m 0.6; Reliability $\hat{p}$ 0.89; Merged from 3 proposals

STEPS

1. Decompose the target object description into a set of required attributes (e.g., size, material, type).
2. Scan the scene specifically for each attribute to identify potential candidates.
3. Verify that a candidate object satisfies all attributes simultaneously before including it in the count.
4. If no object satisfies the full set of attributes, explicitly confirm the count is zero.
5. Identify the target object class and its required attributes.
6. Scan the scene for potential candidates that match the class but fail one or more attributes.
7. Explicitly confirm the count of objects that match the full attribute set.
8. If a category is rare or potentially ambiguous, perform a "negative search" to confirm zero instances exist before concluding the count is zero.

EXAMPLE

To count "large red wooden cubes", first identify all cubes, then filter by size (large), then by color (red), then by material (wood). If any step results in an empty set, the final count is zero.

Consistent unit system selection in mechanics `m_043`

MMLU-Pro Physics; domain `physics`; tags `pitfall.unit_conversion`, `strategy.equation_manipulation`

Retrieved 84; Evidence h_m 6.0; Evidence f_m 1.8; Reliability $\hat{p}$ 0.71

STEPS

1. Identify the unit system used in the problem statement (e.g., CGS vs. SI).
2. Convert all given quantities to the chosen system before performing calculations.
3. Verify that the resulting units for derived quantities (like force or velocity) match the expected output units.
4. Perform a sanity check on the magnitude of the result to ensure no power-of-ten errors occurred during conversion.

EXAMPLE

For a mass of 500 g and a force of 1000 dynes, calculate acceleration in cm/s^2:
$a = F/m = 1000/500 = 2$ cm/s^2.

Solving direct proportionality problems `m_041`

MATH; domain `math.algebra`; tags `strategy.equation_manipulation`, `pitfall.unit_conversion`

Retrieved 126; Evidence h_m 4.1; Evidence f_m 0.0; Reliability $\hat{p}$ 0.84

STEPS

1. Identify the two variables and confirm the relationship is direct ($y = kx$).
2. Use the initial pair of values to calculate the constant of proportionality $k = y/x$.
3. Set up the equation $y = kx$ using the calculated constant.
4. Substitute the new value of x to solve for the target y.
5. Convert decimal values to fractions before performing arithmetic to maintain precision.

EXAMPLE

If 2 cm on a map represents 5 km, how many cm represent 12.5 km? Calculate $k = 2/5 = 0.4$. Then solve $y = 0.4 \times 12.5 = 5$ cm.

E LIMITATIONS AND FUTURE DIRECTIONS

E.1 LIMITATIONS

While LeRe improves accuracy consistently across backbones and domains, it has several limitations, which also suggest directions for future research.

Constrained generation budget. All methods, including LeRe, decode with at most 2,048 output tokens per call (Appendix B). This uniform budget keeps latency and cost low and makes the comparison fair, but it can truncate long chains of reasoning, so backbones whose strength depends on extended reasoning may not reach their full accuracy in our setting. How LeRe's gains change under larger budgets remains to be measured.

Reliance on self-verification. LeRe learns without reference answers, so the quality of its memory depends on the Curator's label-free verdict. Appendix C.2.2 shows that this verdict is informative on every backbone but that its reliability tracks the backbone's capability: the separation between correct and wrong answers, $\zeta_1 - \zeta_0$, falls from 0.31 on Gemini-3.1-flash-lite to 0.14 on GPT-4o-mini, and the verdict is uninformative on HLE-Exact for the two GPT backbones. Weaker backbones therefore supply noisier supervision, which is consistent with LeRe's smallest average gain coming on GPT-4o-mini.

Pre-defined hyperparameters. LeRe has several hyperparameters (Table 4), including the retrieval weights, the consolidation thresholds, the quarantine rule, and the CCME training schedule. They were fixed a priori and shared across all backbones and streams; we examined only the number of retrieved entries K (Appendix C.3.1), so the best setting of the others remains to be determined empirically.

Single-run evaluation. Each accuracy we report is the Pass@1 of a single run. Given the cost and scale of running six methods on twelve streams with three backbones, we did not repeat every setting, which prevents us from reporting error bars or a full statistical significance analysis of Table 1. Decoding at temperature 0 limits run-to-run variation, accuracy changes little across the four retrieval settings of Appendix C.3.1, and preliminary experiments under the same protocol gave consistent results, which suggests that the reported trends are reasonably reliable.

E.2 FUTURE DIRECTIONS

We briefly discuss two directions that follow from these limitations.

Systematic sensitivity analysis. Beyond K, a study of how the retrieval weights, consolidation thresholds, quarantine rule, and CCME schedule interact with the backbone and the stream would show which settings matter and could lead to rules that adapt them online.

Broader learning objectives for retrieval. CCME currently learns from entry-level attributions. A richer objective could also exploit the Solver's reasoning trajectory, for example by crediting the steps in which an entry is actually used, so that the learned geometry aligns memory with the reasoning it helps rather than only with the final outcome of a query.