LEARNING WHAT TO TEACH: ADAPTIVE HINT GENERATION FOR ON-POLICY SELF-DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

On-policy self-distillation (OPSD) turns a language model into its own teacher by supplying it with privileged context (a hint) and distilling the resulting token-level guidance back into the unconditioned student. Existing methods often supply task-level hints that capture what a task generally requires, rather than what the current student gets wrong. We argue that the hint should instead be *learned*. We introduce **EvoHint**, which trains a dedicated hint generator to produce *reflective hints*: guidance conditioned on the student's own attempt and optimized by whether it helps the student succeed. The generator draws on a persistent memory of effective hints, adapting retrieved hints to the observed attempt. Same-task student rollouts provide outcome feedback for generator training and select successful hints for memory updates, allowing guidance to adapt as the student learns. Experiments on agentic tasks (Search-QA and WebShop) and six mathematical reasoning benchmarks show consistent improvements over a fixed-hint self-distillation baseline across student model scales, with gains of up to 9.8 percentage points on agentic tasks and 1.8–3.1 points in average mathematical reasoning accuracy. Ablations further show that EvoHint's gains hold across student objectives, that learned hints outperform frontier-model hints by up to 5.4 points, and that reusing generated hints as memory performs on par with frontier-model-built memory without a separate memory writer.

1 INTRODUCTION

On-policy distillation (OPD) complements reinforcement learning (RL) with dense, token-level supervision: rather than rewarding only the trajectory-level outcome, a teacher scores individual tokens in trajectories sampled from the student's own policy (Agarwal et al., 2024; Shao et al., 2024). On-policy self-distillation (OPSD) eliminates the need for a separate teacher by having the student play this role: conditioned on privileged context, such as a retrieved skill or a reference solution, the model re-scores its own tokens and distills the resulting signal into the unconditioned student (Zhao et al., 2026; Lu et al., 2026; Wang et al., 2026a; Yang et al., 2026a). In this framework, the privileged context, which we call the *hint*, determines the quality of the teaching signal, making the hint the key lever for better distillation.

In existing OPSD methods, hints are fixed before the student attempts the task: retrieved from a library, extracted from prior trajectories, or constructed from reference solutions. Such hints (Hübotter et al., 2026; Yang et al., 2026a; Lu et al., 2026) capture what the task generally requires rather than what the student currently gets wrong, even though a student can fail the same task differently across attempts. Hints are also selected without regard to their downstream effect: guidance that shifts the teacher's token probabilities may fail to improve the student, since teacher-side likelihood changes do not necessarily translate into task success.

We argue that the hint should be *learned* rather than fixed. We propose training a dedicated hint generator that (i) observes the student's own attempt and produces guidance targeted at the mistakes it reveals, and (ii) is optimized for the student's success on subsequent attempts. Grounded in the student's current failures and rewarded by the student's improvement, such a generator naturally adapts as the student evolves.

The student's attempt reveals its specific mistakes, but knowing what went wrong does not by itself indicate how to fix it. In practice, tasks within a domain often share common failure patterns and

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

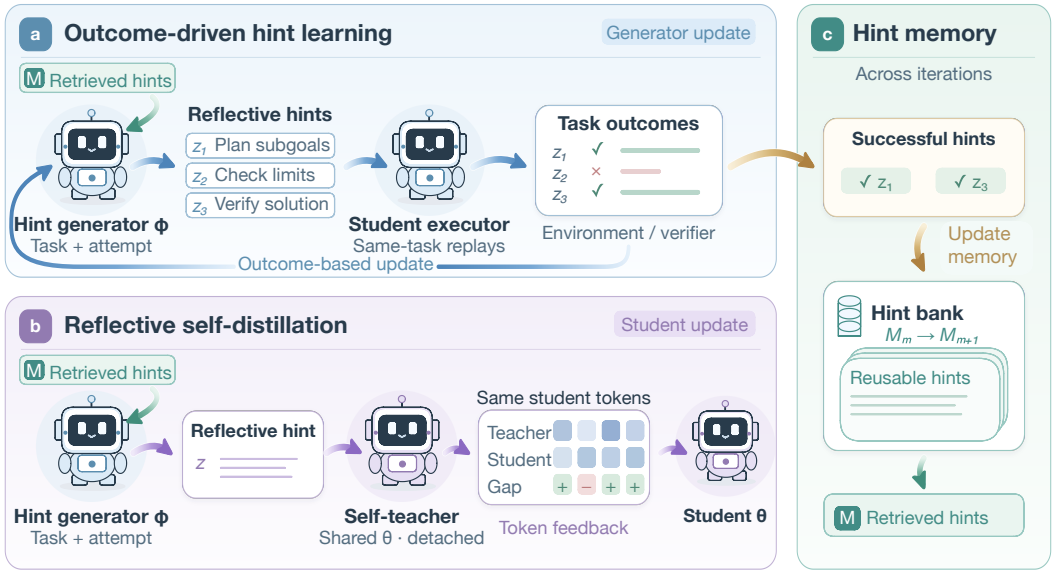


Figure 1: **EvoHint overview.** The student, hint generator, and hint memory co-evolve during training. (a) Same-task rollouts reward useful reflective hints. (b) A detached, hint-conditioned self-teacher scores student tokens for distillation. (c) Successful hints update memory and are retrieved for future hint generation.

effective remedies; a strategy that resolved a particular type of error on one task tends to transfer when the same pattern arises elsewhere. To exploit this structure, we maintain a persistent memory of hints that proved effective across prior tasks and use it to enrich the generator's input: retrieved hints provide grounded, outcome-validated guidance, while the current attempt indicates how that guidance should be adapted to the situation at hand.

We instantiate this idea in **EvoHint** (Figure 1), which augments OPSD with a *hint generator* and a persistent hint memory. Given a task, a sampled student attempt, and hints retrieved from memory, the generator produces a *reflective hint* tailored to the observed attempt; a detached copy of the student model then re-scores the attempt's tokens conditioned on the hint and supplies supervision via a self-distillation objective such as RLSD (Yang et al., 2026a) or SDAR (Lu et al., 2026). The generator is trained with policy-gradient on same-task student rollouts: fresh attempts under each candidate hint yield outcome rewards that directly measure the hint's contribution to task success. Hints that prove effective are written back to memory, where they inform subsequent hint generation on related tasks. The student, hint generator, and memory are updated jointly during training; at inference, the student runs either standalone or with hints retrieved from the frozen memory.

We evaluate EvoHint on two agentic tasks, Search-QA and WebShop, at three model sizes and on six mathematical reasoning benchmarks at two model sizes. EvoHint improves aggregate performance over RLSD baselines at every evaluated scale. Further ablations show gains under both RLSD and SDAR student objectives, and reveal that directly reusing generated hints as memory nearly matches memory built by an external frontier model.

Our contributions are:

- **Learned teacher conditioning.** We propose treating the OPSD hint as a learnable component rather than a fixed artifact, and formalize the objective of training a hint generator to produce attempt-conditioned, outcome-optimized hints.
- **EvoHint.** We introduce a concrete framework that couples the hint generator with a persistent hint memory for cross-task reuse and co-evolves both with the student during training.
- **Empirical evaluation.** We demonstrate consistent improvements over RLSD baselines across two task families (agentic tasks and mathematical reasoning) and multiple model scales, with ablations examining student objectives and memory construction.

2 Related Work

On-policy (self-)distillation. On-policy distillation (OPD) complements outcome-based RL with dense, token-level supervision by having a teacher score tokens sampled from the student's own policy (Agarwal et al., 2024). On-policy self-distillation (OPSD) uses the same model as both teacher and student, conditioned on different contexts (Zhao et al., 2026). A large body of work refines how the teacher signal enters the student update: gating (Lu et al., 2026), advantage reweighting (Yang et al., 2026a), return-associated reward shaping (Zhang et al., 2026), counterfactual credit assignment (Meng & Chen, 2026), recursive Bayesian aggregation (Wang et al., 2026b), and joint adaptation of token selection and privileged-information strength (Yang et al., 2026c). The privileged context that conditions the teacher is typically a retrieved skill (Lu et al., 2026; Wang et al., 2026a), a summary extracted from on-policy trajectories (Yang et al., 2026b), or a reference solution (Zhao et al., 2026); all are fixed before the student attempts the task and matched to the task rather than to the student's specific failures. EvoHint instead treats the conditioning itself as the object to learn.

Learning and evolving teacher conditioning. Several lines of work improve the teacher itself rather than taking it as fixed. RISE (Li et al., 2026b) constructs a synthetic teacher by extrapolating the student's RL training trajectory in parameter or logit space. VPD (Li et al., 2026a) trains the teacher via variational EM to better translate textual feedback into a target token distribution. SEED (Wu et al., 2026) trains the policy itself to generate hindsight skills via SFT on outputs from a stronger LLM, so the skill generator co-evolves with the policy. Memory-augmented methods accumulate conditioning across tasks: UCOB (Tu et al., 2026) maintains a dual-granularity skill memory and trains a skill writer via credit-aware RL; PMD (Liu et al., 2026) co-evolves a three-level procedural memory with the policy. SGSD (Huang et al., 2026) validates skill polarity against verifier outcomes, filtering out conditioning that would harm rather than help the student. These methods show that adapting the teacher during training substantially outperforms fixed retrieval. EvoHint extends this direction in two respects. First, its generator conditions on the student's *current attempt*, producing guidance targeted at the specific failure the attempt reveals rather than at the task in general. Second, it is trained with outcome-based RL on same-task student rollouts, optimizing hint utility for the current student rather than for trajectory summarization or distributional fitting. EvoHint's persistent memory bridges cross-task experience with attempt-specific generation, retrieving proven strategies that the generator adapts rather than applies verbatim.

3 EvoHint

EvoHint learns teacher conditioning through a separately trained hint generator and persistent hint memory (Figure 1). The generator produces reflective hints from student attempts, the student learns from hint-conditioned supervision, and effective hints update memory.

3.1 System Setup

Let $x \sim \mathcal{D}$ be a task and τ a complete attempt generated by the student π_θ, with outcome reward $R(\tau)$. An attempt is either a solution to a reasoning task or an interleaved sequence of actions and observations in an interactive task. For the i-th attempt τ_i, we write $y_{i,t}$ for its t-th response token and $s_{i,t}$ for the preceding context, including environment observations when applicable. Training losses are computed only on generated response tokens. We write $\mathrm{Agg}_{i,t}$ for their masked empirical loss aggregation; sampling contexts and collected responses are held fixed during each optimizer update. Appendix A.1 defines the policy-gradient loss used below.

The *student* π_θ generates hint-free attempts. A detached copy with the same weights acts as the *teacher*, re-scoring those tokens under a hint without receiving gradients. The separately parameterized *hint generator* G_ϕ conditions on the task, student attempt, and retrieved memory to produce a *reflective hint* targeted at the observed mistakes. Thus, distillation scores tokens generated without hints. During generator training, the student also makes fresh hint-conditioned attempts to evaluate candidate hints (Figure 1a). We reuse these trajectories for reward-based GRPO updates, retaining the hint in both the sampling and optimization contexts (Section 3.4); this is RL, not supervised fine-tuning. Only the student and generator receive gradient updates. Persistent hint memory $\mathcal{M}$ supplies context for generation and accumulates effective hints after evaluation.

3.2 PERSISTENT HINT MEMORY

The persistent memory $\mathcal{M}$ stores outcome-selected hints and supplies them as context to the hint generator. For a task x, retrieval returns

$$\mathcal{M}(x) = \mathrm{Retrieve}(\mathcal{M}, x), \tag{1}$$

the (at most k) entries whose embeddings have the highest positive cosine similarity to the task embedding. Retrieved entries are ordered by similarity; retention values are used only for eviction, not for re-ranking. We use $k = 3$ and a capacity of $C = 300$ entries shared across all tasks.

Memory writes reuse the hint-conditioned student rollouts and outcome scores collected during hint-generator training (Section 3.3). Hints that pass the outcome-based acceptance rule are stored without additional student rollouts or outcome evaluations. Each accepted hint is compared with the stored entries by embedding cosine similarity. If its nearest entry exceeds a duplicate threshold of 0.85 (fixed across tasks), the hint reinforces that entry instead of being inserted as a duplicate; otherwise, it is stored as a new entry. At capacity, insertion evicts the entry with the lowest retention value among those not protected by the age-grace rule, falling back to all entries if every entry is protected.

An entry's retention value is $v = n_{\mathrm{helped}}/(n_{\mathrm{retrieved}} + \epsilon)$, where $\epsilon > 0$ is a smoothing constant. The counter $n_{\mathrm{retrieved}}$ increments whenever the entry is retrieved; n_{helped} starts at one and increments whenever a newly accepted hint reinforces the entry. Retention thus measures how often an entry is reinforced by outcome-validated hints relative to how often it is retrieved. Algorithm 3 specifies the full write and eviction procedure.

3.3 HINT GENERATOR UPDATE

The generator is trained to maximize the expected task return of the student when it attempts a task with the generated hint. For a fixed student θ and memory $\mathcal{M}$, the objective is

$$Q_{\theta,\mathcal{M}}(\phi) = \mathbb{E}_{x\sim\mathcal{D},\, \tau'\sim\pi_\theta(\cdot|x),\, z\sim G_\phi(\cdot|\mathcal{M}(x),x,\tau')}\mathbb{E}_{\tau\sim\pi_\theta(\cdot|x,z)}\big[R(\tau)\big], \tag{2}$$

where τ' is a student attempt used to form the hint and τ is a fresh attempt under the generated hint z. For each generator update, the generator selects one hint-free attempt τ' per task and samples H candidate hints $\{z_g\}_{g=1}^{H}$ conditioned on that shared attempt, and for each z_g the student makes K fresh attempts $\tau^{z_g,j} \sim \pi_\theta(\cdot \mid x, z_g)$, which are evaluated by the verifier or environment. The hint score and within-task advantage are

$$\widehat{r}_g = \tfrac{1}{K}\textstyle\sum_{j=1}^{K} R(\tau^{z_g,j}), \qquad A_g^G = \frac{\widehat{r}_g - \bar{r}_x}{\max(\sigma_x, 10^{-6})}, \tag{3}$$

where $\bar{r}_x$ and σ_x are the mean and sample standard deviation of $\{\widehat{r}_g\}_{g=1}^{H}$ for task x. For $H < 2$, we set all hint advantages to zero without evaluating the standard deviation. Following GRPO (Shao et al., 2024), the advantage is broadcast to all tokens of the generated hint, and ϕ is updated with the clipped policy-gradient loss $\mathcal{L}_G = \mathcal{L}_{\mathrm{PG}}(\phi; A^G)$. Identical hint scores give zero advantages. Such groups contribute no policy-gradient surrogate signal; any separately configured regularizer is unaffected. The student and generator parameters remain fixed while these attempts are collected.

3.4 STUDENT POLICY UPDATE

EvoHint is compatible with any OPSD objective that uses hint-conditioned teacher scores; the hint generator and memory operate independently of how those scores enter the student update. With the generator fixed, the student is updated using outcome rewards augmented with teacher scores computed under reflective hints. Hint-free student attempts are sampled from the policy snapshot θ_{old}, and for each response i the generator produces one reflective hint z_i conditioned on the task x, the response itself as the student attempt, and the retrieved hints $\mathcal{M}(x)$. For each sampled token, we compute three log-probabilities, where sg denotes stop-gradient:

$$\ell_{i,t}^{\mathrm{old}} = \log \pi_{\theta_{\mathrm{old}}}(y_{i,t} \mid s_{i,t}), \qquad \ell_{i,t}^{S}(\theta) = \log \pi_\theta(y_{i,t} \mid s_{i,t}),$$
$$\ell_{i,t}^{T} = \mathrm{sg}[\log \pi_{\theta_{\mathrm{old}}}(y_{i,t} \mid s_{i,t}, z_i)]\,. \tag{4}$$

The teacher ($\ell_{i,t}^{T}$) and old-policy ($\ell_{i,t}^{\mathrm{old}}$) scores are cached, while $\ell_{i,t}^{S}(\theta)$ is recomputed as θ is updated. Let A_i denote the fixed outcome-reward advantage for response i, broadcast to its generated tokens,

and let A collect these advantages used in the base policy-gradient loss $\mathcal{L}_{\mathrm{PG}}(\theta; A)$. We instantiate EvoHint with two representative objectives: RLSD and SDAR.

SDAR-style gated supervision. With σ denoting the sigmoid function,

$$g_{i,t} = \mathrm{sg}\big[\sigma\big(\beta\,(\ell^T_{i,t} - \ell^S_{i,t}(\theta))\big)\big]\,, \qquad \mathcal{L}_{\mathrm{SDAR}} = \mathrm{Agg}_{i,t}\big[g_{i,t}\,(\ell^T_{i,t} - \ell^S_{i,t}(\theta))\big]\,,$$
$$\mathcal{L}_S = \mathcal{L}_{\mathrm{PG}}(\theta; A) + \alpha\mathcal{L}_{\mathrm{SDAR}}. \tag{5}$$

Following Lu et al. (2026), we use $\alpha = 0.01$ and $\beta = 5$. Both the gate and teacher score are detached, so the auxiliary gradient on the fixed batch is $-\mathrm{Agg}_{i,t}[g_{i,t}\nabla_\theta \ell^S_{i,t}]$. This is a gated sampled-token surrogate, not a nonnegative KL divergence.

RLSD-style advantage reweighting. RLSD uses the teacher–old-policy difference $\delta_{i,t} = \ell^T_{i,t} - \ell^{\mathrm{old}}_{i,t}$ to reweight the reward advantage:

$$w_{i,t} = \mathrm{clip}(\exp(\mathrm{sign}(A_i)\,\delta_{i,t}),\ 1-\epsilon_w,\ 1+\epsilon_w)\,, \qquad \lambda_s = \lambda_0 \max(0, 1 - s/W),$$
$$\widetilde{A}_{i,t} = A_i\big[(1-\lambda_s) + \lambda_s\, w_{i,t}\big], \qquad \mathcal{L}_{\mathrm{RLSD}} = \mathcal{L}_{\mathrm{PG}}(\theta;\, \widetilde{A}). \tag{6}$$

Here $s \geq 0$ is the training step; the cached scores and $\widetilde{A}$ remain fixed during the student update. Our configuration uses $\lambda_0 = 0.5$, $W = 100$, and $\epsilon_w = 0.2$. The multiplier $(1-\lambda_s) + \lambda_s w_{i,t}$ lies in $[1-\lambda_s\epsilon_w,\ 1+\lambda_s\epsilon_w]$ and is strictly positive for these settings, so it never reverses the sign of the advantage; the distillation contribution decays linearly to zero by step W.

When hinted rollouts are reused, the losses above apply to the plain rows. Hinted rows instead contribute $\mathcal{L}_{\mathrm{PG}}(\theta; A^{\mathrm{hint}})$ with their hint-conditioned sampling contexts and ordinary reward advantages. The combined loss aggregates both sets of valid tokens; no additional SDAR term or RLSD reweighting is applied to the hinted rows (Appendix A.4).

3.5 Training Loop

EvoHint alternates student and generator updates, holding the other policy fixed. At iteration m, the student samples N hint-free rollouts per task. The generator conditions on each attempt and retrieved memory $\mathcal{M}_m(x)$ to produce reflective hints for teacher scoring. For generator learning, it samples H candidate hints per task and collects K fresh student attempts per hint; outcome advantages in Eq. (3) update ϕ, and accepted hints update memory to $\mathcal{M}_{m+1}$. With rollout reuse, generator evaluation precedes the student optimizer step so the retained hinted attempts can join its training batch, preserving their sampling-time probabilities and hint contexts. Without reuse, the student optimizer step precedes generator evaluation (Algorithm 1).

Inference. EvoHint supports two deployment modes. In the *standalone* mode, the student answers test tasks from the prompt alone, and the hint generator, teacher, and memory are discarded. In the *memory-retrieval* mode, the memory is frozen at the end of training: for each test task, the most similar stored hint is retrieved and prepended to the prompt, reusing effective hints accumulated during training without requiring the generator.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate EvoHint on two multi-turn agentic benchmarks and six mathematical reasoning benchmarks. *Search-based QA* (Lu et al., 2026) covers seven search-augmented datasets: NQ, TriviaQA, PopQA, HotpotQA, 2WikiMultiHopQA, MuSiQue, and Bamboogle. Following Lu et al. (2026); Feng et al. (2025), we train on NQ and HotpotQA and evaluate across all seven; we report per-dataset accuracy and the unweighted seven-dataset average. *WebShop* (Yao et al., 2022) is a web-based product-selection environment; we train on 1000 tasks and evaluate on a fixed 500-task held-out set, reporting graded task Score and success rate (Acc.). For *mathematical reasoning*, we train on DAPOMath (Yu et al., 2026) and evaluate on MATH500, AIME24, AIME25, AMC23, Minerva, and OlympiadBench, reporting per-benchmark accuracy and the unweighted six-benchmark average.

Models and configuration. For agentic tasks, we use Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, and Qwen3-1.7B; for mathematical reasoning, we use Qwen3-1.7B and Qwen3-8B. In all main

Table 1: **Agent benchmark results.** Search-QA accuracy and WebShop performance (%; higher is better), grouped by student model. Starred entries retain skills at inference.

	Search-QA								WebShop	
Method	NQ	Trivia	PopQA	Hotpot	2Wiki	MuS.	Bam.	Avg.	Score	Acc.
Qwen2.5-3B-Instruct										
Base	24.6	48.1	31.0	26.3	25.3	7.2	59.7	31.7	6.7	0.8
Skill-Prompt*	23.7	46.2	30.6	24.4	22.1	7.5	12.5	23.9	0.2	0.8
OPSD	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	11.3	3.1
GRPO	39.3	60.6	41.1	37.4	34.6	15.4	26.4	36.4	79.8	63.3
Skill-GRPO	43.5	58.8	43.0	36.8	32.2	11.7	12.5	34.1	77.3	60.9
Skill-GRPO*	44.3	59.6	44.3	39.0	36.1	14.5	14.9	36.1	76.3	66.4
GRPO+OPSD	**44.9**	**61.2**	45.2	**40.4**	38.5	16.0	66.1	**44.6**	77.8	66.4
Skill-SD	44.4	60.4	44.0	39.5	**40.4**	15.4	64.9	44.1	75.9	64.0
RLSD	41.5	58.6	42.3	**40.4**	40.2	**16.8**	**66.9**	43.8	84.4	66.4
SDAR	44.8	58.1	44.3	38.6	36.2	15.7	66.1	43.4	85.0	68.0
EvoHint (ours)	44.6	61.0	**45.3**	39.1	40.3	14.3	**66.9**	44.5	**87.2**	**69.6**
Qwen2.5-7B-Instruct										
Base	25.2	50.8	29.5	29.0	29.0	10.4	63.7	33.9	5.9	1.6
Skill-Prompt*	30.9	52.1	32.7	32.7	27.9	12.7	66.1	36.4	1.7	0.8
OPSD	8.8	8.6	17.5	2.5	4.2	0.5	1.2	6.2	4.5	2.3
GRPO	45.1	63.7	44.0	43.6	43.2	16.8	37.6	42.0	80.9	72.6
Skill-GRPO	45.2	63.7	45.7	43.1	43.3	19.6	21.4	40.3	80.4	71.9
Skill-GRPO*	44.8	63.0	45.1	43.7	43.7	20.5	71.4	47.5	87.0	81.2
GRPO+OPSD	47.3	**64.5**	46.9	43.8	39.3	18.0	69.4	47.0	86.8	76.5
Skill-SD	47.1	**64.5**	47.8	44.2	42.1	20.2	69.0	47.8	86.1	76.5
RLSD	46.8	63.0	44.4	45.5	48.9	**21.5**	73.0	49.0	87.4	77.3
SDAR	46.3	63.5	48.2	43.8	48.4	19.6	73.0	49.0	89.4	**82.8**
EvoHint (ours)	**47.6**	63.8	**49.2**	**45.7**	**49.2**	21.3	**73.2**	**50.0**	**91.1**	80.2
Qwen3-1.7B										
Base	29.4	46.9	37.0	23.5	19.6	6.4	10.5	24.8	46.5	4.7
Skill-Prompt*	29.4	46.5	36.2	22.9	20.8	4.3	10.1	24.3	23.0	2.3
OPSD	4.2	8.3	4.6	6.6	15.3	0.7	1.2	5.8	47.4	9.3
GRPO	40.0	58.9	43.5	35.4	30.3	12.0	65.7	40.8	67.3	38.3
Skill-GRPO	39.2	58.6	43.9	35.2	28.2	11.5	66.1	40.4	73.4	46.1
Skill-GRPO*	38.0	58.4	43.9	36.3	29.0	12.5	66.9	40.7	80.4	50.0
GRPO+OPSD	40.7	58.9	45.0	37.0	34.6	**13.3**	65.7	42.2	70.7	38.3
Skill-SD	39.1	57.5	45.4	34.8	34.1	10.7	64.1	40.8	81.8	53.9
RLSD	38.6	57.3	43.0	34.5	34.1	11.5	65.3	40.6	75.8	56.8
SDAR	39.7	58.9	45.3	35.9	35.5	12.6	65.3	41.9	76.8	58.6
EvoHint (ours)	**43.6**	**60.8**	**47.2**	**38.0**	**36.0**	12.8	**66.5**	**43.6**	**85.6**	**63.2**

experiments, the hint generator uses the same model variant and parameter size as the corresponding student, but is instantiated as a separate model with independently trained parameters. Unlike the self-teacher, it does not share weights with the student. Main-table results use EvoHint with the RLSD student objective. The strategy memory is initialized with the SkillBank from SkillRL (Xia et al., 2026) and updated throughout training as effective hints are accumulated. Unless otherwise noted, the student is evaluated in standalone mode (without hints or memory during inference).

Baselines. We compare against methods spanning outcome-only RL, privileged self-distillation, and hybrid objectives. *Base* reports performance before task-specific training. *GRPO* (Shao et al., 2024) optimizes task rewards without teacher feedback. For agentic tasks, *Skill-Prompt* retrieves skills only at inference, and *Skill-GRPO* incorporates them during RL training; both use the SkillBank from SkillRL (Xia et al., 2026), following Lu et al. (2026). On the distillation side, *OPSD* (Zhao et al., 2026) learns from a skill-conditioned self-teacher without outcome-based RL, while *GRPO+OPSD* adds that distillation loss to GRPO. *Skill-SD* (Wang et al., 2026a) schedules skill-conditioned distillation, *SDAR* (Lu et al., 2026) gates token-level auxiliary supervision, and *RLSD* (Yang et al., 2026a) reweights reward-derived token advantages using teacher feedback. For mathematical reasoning, *GRPO+SDPO*, *SDAR*, and *RLSD* condition the teacher on a correct solution sampled for the same problem (Hübotter et al., 2026). RLSD is the closest comparison to EvoHint: both use the same student update, but RLSD relies on fixed teacher conditioning whereas EvoHint learns the hint with a trained generator.

Table 2: **Mathematical reasoning.** Accuracy (%; higher is better) on six benchmarks.

	Mathematical Reasoning						
Method	MATH500	AIME24	AIME25	AMC23	Minerva	OlyBench	Avg.
Qwen3-8B							
Base	73.2	27.1	23.1	66.1	21.1	48.6	43.2
GRPO	83.8	54.4	42.9	89.4	31.5	57.9	60.0
GRPO+SDPO	83.2	42.3	32.7	84.4	30.6	62.5	55.9
SDAR	**84.5**	50.2	38.3	89.4	31.2	**64.7**	59.7
RLSD	83.0	53.1	40.0	90.6	31.3	60.5	59.8
EvoHint (ours)	84.2	**55.1**	**43.6**	**91.2**	**32.3**	63.0	**61.6**
Qwen3-1.7B							
Base	64.6	11.5	12.1	41.9	17.6	40.3	31.3
GRPO	75.6	**30.0**	26.3	65.6	23.9	51.1	45.4
GRPO+SDPO	76.0	21.5	23.1	65.0	23.1	50.3	43.2
SDAR	75.0	26.0	26.0	**71.3**	23.6	50.8	45.5
RLSD	75.3	26.7	25.2	59.4	23.0	49.8	43.2
EvoHint (ours)	**76.6**	27.8	**29.2**	66.2	**25.2**	**52.8**	**46.3**

4.2 Main Results

Agentic tasks. EvoHint improves over RLSD at all three model sizes on Search-QA and WebShop (Table 1), achieving the highest WebShop Score among listed methods at each size. Gains are largest for Qwen3-1.7B, the strongest hint-free method overall: Search-QA accuracy rises from 40.6% to 43.6% and WebShop accuracy from 56.8% to 63.2%. EvoHint leads or ties RLSD on most individual Search-QA datasets; at 7B, it improves six of seven and achieves the highest average among listed methods. The benefits extend across scales but vary by metric: SDAR retains higher WebShop accuracy at 7B, while GRPO+OPSD has a slightly higher Search-QA average at 3B.

Mathematical reasoning. EvoHint achieves the highest six-benchmark average at both sizes: 46.3% for Qwen3-1.7B and 61.6% for Qwen3-8B (Table 2), exceeding RLSD by 3.1 and 1.8 percentage points. It improves over RLSD on all six benchmarks and leads the listed methods on four at each size. At 1.7B, the largest gains over RLSD occur on AMC23 ($+6.8$ points) and AIME25 ($+4.0$); at 8B, they occur on AIME25 ($+3.6$) and OlympiadBench ($+2.5$). The larger average gain at 1.7B is consistent with the stronger improvements for this model on the agentic tasks.

5 Ablation Studies

We examine frontier models for hint generation and memory creation, memory retrieval at inference, hint conditioning and student objectives, and update schedules and rollout count.

5.1 Frontier Models for Hint and Memory

Table 3 examines the use of a frontier model in two roles: generating hints and creating memory entries. The comparisons use Qwen3-1.7B on Search-QA and WebShop with the RLSD student objective. The EvoHint variants retain persistent memory, while RLSD without co-evolution provides the baseline reference.

Table 3: **Hint and memory alternatives.** Qwen3-1.7B with RLSD on WebShop (%).

Hint	Memory	Score	Acc.
EvoHint	Direct hint reuse	85.6	63.2
GPT-5.5	Direct hint reuse	80.2	60.4
EvoHint	GPT-5.5	**85.8**	**64.6**
EvoHint	Student policy	72.2	55.6

Hint generation. On WebShop, our learned hint generator exceeds GPT-5.5 by 5.4 Score points and 2.8 accuracy points. On Search-QA, it improves the seven-dataset average from 42.5% to 43.6%, with higher accuracy on five of seven datasets; GPT-5.5 retains small advantages on 2Wiki and MuSiQue. These comparisons favor the learned generator over fixed frontier-model hints, consistent with the motivation for adapting guidance to the student's attempts and task outcomes.

Memory creation. On WebShop, using GPT-5.5 to create memory entries improves Score by 0.2 points and accuracy by 1.4 points over directly storing the hint generator's output. On Search-QA, GPT-5.5 memory creation raises the average from 43.6% to 43.8%, with gains on HotpotQA, 2Wiki, and MuSiQue offset by lower accuracy on the other four datasets. Direct reuse therefore remains competitive with frontier-model memory creation while avoiding a separate call to generate each entry. Task feedback selects which generated hints are stored, preserving guidance that has already supported successful student attempts.

5.2 Memory at Inference

Table 4 compares the same trained student with and without frozen-memory retrieval at inference. Both settings use persistent memory during training; only its use at inference changes. On WebShop, retrieval improves Score by 1.6 points to 87.2 and accuracy by 3.6 points to 66.8%. On Search-QA, average accuracy drops from 43.6% to 41.3%, with six of seven datasets declining. Thus, the hints accumulated during training provide useful inference-time guidance on WebShop, but do not improve performance uniformly across tasks.

One possible explanation is that WebShop benefits from reusable guidance on intermediate decisions, for which sparse outcome rewards provide limited feedback. In Search-QA, success depends more on evidence specific to the current question; embedding similarity can retrieve a hint whose factual assumptions or search direction do not fit that question. Such mismatched guidance may distract the student, suggesting that retrieval relevance alone does not ensure useful guidance.

Table 4: **EvoHint inference modes.** Qwen3-1.7B with RLSD (%).

	Search-QA								WebShop	
EvoHint inference mode	NQ	Trivia	PopQA	Hotpot	2Wiki	MuS.	Bam.	Avg.	Score	Acc.
w/o memory in inference	**43.6**	**60.8**	**47.2**	**38.0**	**36.0**	12.8	**66.5**	**43.6**	85.6	63.2
w/ memory in inference	39.6	57.3	42.4	36.5	35.4	**12.9**	65.3	41.3	**87.2**	**66.8**

5.3 Hint Conditioning and Student Objectives

Hint conditioning. Reflective hints use the task, a student attempt, and retrieved memory; proactive hints omit the attempt. Table 5 compares three configurations: reflective hints in both update stages (our main method), proactive hints in both stages, and reflective hints for generator learning with proactive hints for student learning. All three use Qwen3-1.7B on WebShop with the RLSD student objective, persistent memory, and direct hint reuse for memory creation. The mixed configuration lets us examine attempt conditioning in generator learning and teacher scoring separately.

Table 5: **Hint conditioning across update stages.** Qwen3-1.7B on WebShop (%).

Generator update	Student update	Score	Acc.
RLSD (no co-evolution)		75.8	56.8
Reflective	Reflective (main)	**85.6**	**63.2**
Proactive	Proactive	74.8	57.6
Reflective	Proactive	78.2	59.2

Using proactive hints in both stages yields a score of 74.8 and an accuracy of 57.6%, close to the RLSD baseline without co-evolution (75.8 / 56.8%). Switching only generator learning to reflective hints raises score to 78.2 and accuracy to 59.2%, gains of 3.4 score points and 1.6 percentage points. Using reflective hints in the student update as well further improves these metrics to 85.6 and 63.2%, adding 7.4 Score points and 4.0 percentage points. Reflective conditioning in both stages performs best, with the larger incremental gain in the student update.

Student objectives. Figure 2 compares SDAR (Lu et al., 2026) and RLSD (Yang et al., 2026a) with and without EvoHint. Both objectives benefit: Score improves by 4.4 points for SDAR and 9.8 for RLSD, while accuracy rises by 3.2 and 6.4 percentage points, respectively. These gains support using learned teacher conditioning with both gated supervision and advantage reweighting.

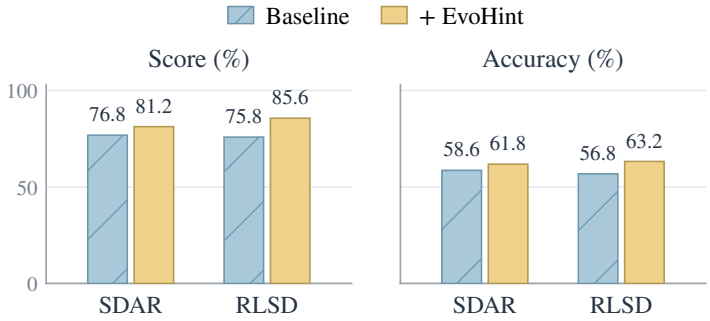


Figure 2: **EvoHint across student objectives.** Qwen3-1.7B on WebShop.

5.4 Update Schedule and Rollout Count

Table 6: **Update schedule and rollout count.** Qwen3-1.7B on WebShop (%).

(a) Update frequency.

n_S	n_G	Score	Acc.
20	10	71.8	50.1
20	5	72.6	52.0
10	5	73.2	52.4
1	1	**85.6**	**63.2**

(b) Rollouts per hint.

K	Score	Acc.
1	79.8	60.2
2	**81.2**	**61.8**
4	80.6	61.0

Update frequency. Table 6(a) varies student and generator update counts n_S and n_G. Updating each policy once per iteration exceeds the strongest block schedule, $(10, 5)$, by 12.4 Score points and 10.8 accuracy percentage points. Shortening (n_S, n_G) from $(20, 10)$ to $(10, 5)$ also improves both metrics at a fixed 2:1 ratio. These results suggest benefits from frequent feedback, which may keep the two policies better matched as they evolve. Compared with the default, these schedules change both the update frequency and the student-to-generator update ratio.

Rollout count. Table 6(b) varies the number of student attempts K used to estimate each hint's outcome reward with SDAR. Increasing K from 1 to 2 improves Score by 1.4 points and accuracy by 1.6 percentage points. The second attempt reduces reliance on a single trajectory when scoring a hint. Increasing K to 4 doubles the hinted rollout count without further gains. Thus, $K = 2$ offers the best observed tradeoff between performance and hinted rollout count.

6 Conclusion

EvoHint learns reflective teacher-conditioning hints from student attempts and task outcomes, co-evolving its generator, student, and persistent memory. Experiments show gains across interactive agents, mathematical reasoning, and student objectives. Learned hints outperform frontier-model hints, while direct hint reuse remains competitive with frontier-model memory creation. Ablations favor reflective conditioning in both update stages. The trained student supports standalone inference or frozen-memory retrieval without the generator. Retrieval improves WebShop performance but reduces Search-QA accuracy, highlighting the need to adapt memory use to the task. Future work could learn when to retrieve and explore reusing the generator at inference.

AI use statement

We used generative AI tools to assist with code development and with drafting, polishing, and formatting the manuscript, including figures and tables. All AI-assisted text and figures were reviewed, revised, and verified by the authors. AI-assisted code was reviewed and tested for correctness. We take responsibility for the final content of this work, including all text, claims, code, and artifacts.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5be69a584901a26c521c2b51e40a4c20-Abstract-Conference.html.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. *arXiv preprint arXiv:2505.10978*, 2025.

Jiazhen Huang, Xiao Chen, Xiao Luo, Yong Dai, Senkang Hu, and Yuzhi Zhao. Skill-conditioned gated self-distillation for llm reasoning. *arXiv preprint arXiv:2605.28791*, 2026.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Yang Li, Erik Nijkamp, Semih Yavuz, and Shafiq Joty. Learning from language feedback via variational policy distillation. *arXiv preprint arXiv:2605.15113*, 2026a.

Yang Li, Semih Yavuz, and Shafiq Joty. RISE: Recursive improvement via self-extrapolating policy distillation. *arXiv preprint arXiv:2609.05295*, 2026b. URL https://arxiv.org/abs/2609.05295.

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

Zhengxi Lu, Zhiyuan Yao, Zhuowen Han, Zi-Han Wang, Jinyang Wu, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Self-distilled agentic reinforcement learning. *arXiv preprint arXiv:2605.15155*, 2026.

Zibin Meng and Kani Chen. Craft: Counterfactual credit assignment from free sibling rollouts for self-distilled agentic reinforcement learning. *arXiv preprint arXiv:2606.29476*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Songjun Tu, Chengdong Xu, Qichao Zhang, Yiwen Ma, Yaocheng Zhang, Linjing Li, Dong Li, Xiangyuan Lan, and Dongbin Zhao. UCOB: Learning to utilize and evolve agentic skills via credit-aware on-policy bidirectional self-distillation. *arXiv preprint arXiv:2606.29502*, 2026.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. Skill-SD: Skill-conditioned self-distillation for multi-turn LLM agents. *arXiv preprint arXiv:2604.10674*, 2026a.

Zi-Han Wang, Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Jie Wu, Zhengzhou Cai, Yueqing Sun, Ziang Ye, Linji Hao, Qi Gu, et al. Agentopsd: Recursive self-distillation for agentic reinforcement learning. *arXiv preprint arXiv:2608.05987*, 2026b.

Jinyang Wu, Shuo Yang, Zhengxi Lu, Fan Zhang, Yuhao Shen, Lang Feng, Haoran Luo, Zheng Lian, Shuai Zhang, Zhengqi Wen, et al. Seed: Self-evolving on-policy distillation for agentic reinforcement learning. *arXiv preprint arXiv:2607.14777*, 2026.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled RLVR. *arXiv preprint arXiv:2604.03128*, 2026a.

Shuo Yang, Jinyang Wu, Zhengxi Lu, Yuhao Shen, Fan Zhang, Lang Feng, Shuai Zhang, Haoran Luo, Zheng Lian, Zhengqi Wen, and Jianhua Tao. OPID: On-policy skill distillation for agentic reinforcement learning. *arXiv preprint arXiv:2606.26790*, 2026b.

Yongkang Yang, Zhezheng Hao, Hong Zhang, Yi Liu, Xiankun Lin, Wence Ji, Fanjunduo Wei, Jiarui Yu, Qiang Lin, Xiaoyun Liang, et al. Matching supervision to the student's learning capacity: A unified framework for on-policy self-distillation. *arXiv preprint arXiv:2608.08176*, 2026c.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. *arXiv preprint arXiv:2207.01206*, 2022. URL https://arxiv.org/abs/2207.01206.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Ranxu Zhang, Guinan Chen, Jinghao Lin, Xiaozhou Xu, Yanyong Zhang, Chao Wang, et al. Agentic reinforcement learning with self-distilled reward shaping. *arXiv preprint arXiv:2608.03223*, 2026.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026. URL https://arxiv.org/abs/2601.18734.

APPENDIX CONTENTS

A Algorithms **13**

A.1 Policy-Gradient Loss Used by Both Policies 13

A.2 EvoHint Training 14

A.3 Generator Learning and Memory Updates 15

A.4 Student Updates: SDAR and RLSD 16

A.5 Baseline Training Algorithms 17

A ALGORITHMS

We give the complete EvoHint procedure, followed by its generator and memory updates and the two student-update interfaces. Baseline algorithms use the same notation to make their supervision sources explicit. All losses aggregate only valid response tokens. A rollout denotes either a mathematical solution or an interactive trajectory.

A.1 POLICY-GRADIENT LOSS USED BY BOTH POLICIES

We use $\mathcal{L}_{\mathrm{PG}}(\psi; A)$ to denote a clipped policy-gradient loss for the policy being updated. Let q_ψ generate token $u_{i,t}$ under context $d_{i,t}$, and let ψ_{old} denote its sampling-time weights. The probability ratio is

$$r_{i,t}(\psi) = \frac{q_\psi(u_{i,t} \mid d_{i,t})}{q_{\psi_{\mathrm{old}}}(u_{i,t} \mid d_{i,t})}. \tag{7}$$

For token advantages $A_{i,t}$, the standard clipped surrogate is

$$\mathcal{L}_{\mathrm{PG}}(\psi; A) = -\operatorname{Agg}_{i,t}[\min\{r_{i,t}(\psi)A_{i,t}, \operatorname{clip}(r_{i,t}(\psi), 1-\epsilon_-, 1+\epsilon_+)A_{i,t}\}]\,. \tag{8}$$

The aggregation $\operatorname{Agg}_{i,t}$ is the configured masked empirical average over valid response tokens. For token-mean reduction, $\operatorname{Agg}[f] = \sum_{i,t} m_{i,t} f_{i,t} / \sum_{i,t} m_{i,t}$, where $m_{i,t}$ is the response mask. The sampled tokens, contexts, old log probabilities, advantages, and masks are fixed during differentiation. The displayed loss is minimized. The archived implementation also applies dual clipping: for $A_{i,t} < 0$, the quantity inside the aggregation is lower-bounded by $cA_{i,t}$ with $c > 1$. Loss aggregation, clipping thresholds, and any additional entropy or reference-policy regularization follow the training configuration and are separate from the displayed surrogate. Outcome-based group advantages follow GRPO (Shao et al., 2024).

For the **student**, $q_\psi = \pi_\theta$, $u_{i,t} = y_{i,t}$, and $d_{i,t} = s_{i,t}$ for plain rows and $d_{i,t} = (s_{i,t}, z_i)$ for reused hinted rows. The hint remains in both the numerator and denominator contexts for hinted rows. On plain rows, SDAR retains the reward-derived advantages and adds its gated auxiliary; RLSD supplies $\widetilde{A}_{i,t}$ from Eq. (6). For the **generator**, $q_\psi = G_\phi$, the generated tokens belong to a reflective hint, and the conditioning context contains the retrieved memory, attempt context, and preceding hint tokens. Each hint's advantage A_g^G from Eq. (3) is broadcast to its tokens. Thus generator probability ratios use $G_\phi / G_{\phi_{\mathrm{old}}}$; student ratios use $\pi_\theta / \pi_{\theta_{\mathrm{old}}}$. Neither ratio compares teacher scores to student scores.

A.2 EvoHint Training

Algorithm 1 shows the schedule with both policies updated in each iteration. Both stages generate reflective hints from a task, a student attempt, and retrieved memory. $\mathrm{StudentUpdate}$ uses Algorithm 4 or 5. Replay reuse supplies the hinted trajectories to this update in addition to the original plain trajectories.

Algorithm 1 EvoHint: reflective co-evolution with persistent memory

Input: Student π_θ, generator G_ϕ, memory $\mathcal{M}_0$, task distribution $\mathcal{D}$; rollout counts N, H, K; student objective; replay-reuse flag u
Output: Trained student π_θ and memory $\mathcal{M}_T$

for iteration $m = 0, \ldots, T-1$ **do**
  Sample task batch $\mathcal{B}_m$; snapshot $\theta_{\mathrm{old}} \leftarrow \theta$
  Initialize plain trajectory batch $\mathcal{P} \leftarrow \varnothing$
  for task $x \in \mathcal{B}_m$ **do**
    Retrieve up to three entries by positive cosine similarity; set $b_m(x) \leftarrow \mathcal{M}_m(x)$
    Update retrieval counts and entry values
    Sample N plain attempts with $\pi_{\theta_{\mathrm{old}}}(\cdot \mid x)$ and evaluate rewards
    Form task-and-attempt contexts $c^{\mathrm{att}} = (x, \tau')$ from these attempts
    Generate reflective scoring hints with $G_\phi(\cdot \mid b_m(x), c^{\mathrm{att}})$
    Cache old-policy and detached teacher scores using Eq. (4)
    Add attempts, rewards, contexts, and cached scores to $\mathcal{P}$
  end for
  if $u = 0$ **then**
    $\theta \leftarrow \textsc{StudentUpdate}(\theta, \mathcal{P}, \varnothing)$
  end if
  Freeze the current student as the executor for this generator update
  $(\phi, \mathcal{H}, \mathcal{R}) \leftarrow \textsc{GeneratorUpdate}(\phi, \theta, \mathcal{B}_m, b_m, c^{\mathrm{att}}, H, K)$
  $\mathcal{M}_{m+1} \leftarrow \textsc{MemoryUpdate}(\mathcal{M}_m, \mathcal{H})$
  if $u = 1$ **then**
    $\theta \leftarrow \textsc{StudentUpdate}(\theta, \mathcal{P}, \mathcal{R})$
  end if
end for
return $(\pi_\theta, \mathcal{M}_T)$ ▷ Standalone or frozen-memory inference

The student update precedes generator learning when replay reuse is disabled. With reuse, generator evaluation comes first so its trajectories are available to the student update. Executor weights remain fixed during each generator evaluation. Without reuse, the attempt contexts come from the pre-update student, so this evaluation uses the collected context distribution rather than resampling contexts from the updated executor. Sampling-time weights and detached scores are refreshed for each new batch.

A.3 Generator Learning and Memory Updates

Algorithm 2 evaluates hints through fresh attempts on the same task. $\mathcal{H}$ records candidate hints and outcomes for the acceptance rule; $\mathcal{R}$ retains the hinted trajectories and their sampling-time log probabilities for possible reuse.

Algorithm 2 GeneratorUpdate: outcome-based reflective hint learning

Input: Generator G_ϕ, fixed executor π_θ, tasks $\mathcal{B}$, retrieved memory b, attempt contexts c^{att}, counts H, K
Output: Updated ϕ, hint records $\mathcal{H}$, replay batch $\mathcal{R}$

Snapshot $\phi_{\mathrm{old}} \leftarrow \phi$; initialize $\mathcal{H}, \mathcal{R} \leftarrow \varnothing$
for task $x \in \mathcal{B}$ **do**
  Select a shared attempt context c_x^{att}
  Sample $z_{1:H} \sim G_{\phi_{\mathrm{old}}}(\cdot \mid b(x), c_x^{\mathrm{att}})$; cache hint log probabilities
  for $g = 1, \ldots, H$ **do**
    Generate K fresh attempts $\tau^{z_g,k} \sim \pi_\theta(\cdot \mid x, z_g)$
    Evaluate $R(\tau^{z_g,k})$; compute $\widehat{r}_g = K^{-1} \sum_k R(\tau^{z_g,k})$
    Retain the trajectories selected for reuse, their rewards, hinted contexts, and old log probabilities in $\mathcal{R}$
    Append $(x, c_x^{\mathrm{att}}, z_g, \widehat{r}_g)$ and acceptance-rule inputs to $\mathcal{H}$
  end for
  Compute A_g^G across hints using Eq. (3)
  For $H < 2$, set $A_g^G = 0$ without computing a sample standard deviation; broadcast to hint tokens
end for
Update ϕ with $\mathcal{L}_{\mathrm{PG}}(\phi; A^G)$, using ratios against $G_{\phi_{\mathrm{old}}}$
return $(\phi, \mathcal{H}, \mathcal{R})$

Algorithm 3 MemoryUpdate: outcome-selected hint retention

Input: Memory $\mathcal{M}$, evaluated hints $\mathcal{H}$, acceptance rule a_m, capacity $C = 300$, duplicate threshold $d = 0.85$, smoothing $\epsilon > 0$, age-grace rule
Output: Updated memory $\mathcal{M}'$

$\mathcal{M}' \leftarrow \mathcal{M}$
for candidate record $h = (x, c_x^{\mathrm{att}}, z, \widehat{r}, \ldots) \in \mathcal{H}$ **do**
  if $a_m(h) = 1$ and z is nonempty **then**
    Embed z; find the stored hint entry e with highest cosine similarity, if one exists
    if e exists and similarity exceeds d **then**
      Increment $n_{\mathrm{helped}}(e)$
    else
      if $|\mathcal{M}'| = C$ **then**
        Evict the lowest-value age-eligible entry; use all entries if none is eligible
      end if
      Insert z with $n_{\mathrm{helped}} = 1$, $n_{\mathrm{retrieved}} = 0$, and its insertion time
    end if
    Refresh affected values $v(e) = n_{\mathrm{helped}}(e)/(n_{\mathrm{retrieved}}(e) + \epsilon)$
  end if
end for
return $\mathcal{M}'$

The acceptance rule is the outcome-based gate described in Section 3.2; its threshold is a configuration choice. Retrieval increments usage counts, while accepted duplicate hints reinforce existing entries. All EvoHint variants retain persistent memory.

A.4 STUDENT UPDATES: SDAR AND RLSD

The inputs below are a plain on-policy batch $\mathcal{P}$ with cached scores from Eq. (4), reward-derived advantages A, and an optional replay batch $\mathcal{R}$. EvoHint supplies reflective hints to the scoring teacher. For the corresponding baseline, the teacher context comes from that baseline's fixed guidance procedure. The teacher scores remain detached throughout the update.

We write $\mathrm{Combine}$ for aggregation over concatenated plain and replay rows using the configured token masks and loss reduction. When $\mathcal{R} = \varnothing$, set $\mathcal{L}_{\mathrm{replay}} = 0$ and return the plain loss. Otherwise, the replay rows contribute ordinary policy-gradient loss using their hint-conditioned sampling contexts and sampling-time probabilities. They receive neither the gated auxiliary nor RLSD advantage reweighting.

Algorithm 4 StudentUpdate with SDAR-style gated supervision

Input: θ, plain batch $\mathcal{P}$, replay batch $\mathcal{R}$, coefficients α, β
Output: Updated student parameters θ

Compute reward-derived advantages for $\mathcal{P}$ and, if present, $\mathcal{R}$
for optimization minibatch **do**
  Recompute $\ell^S_{i,t}(\theta)$ on plain student tokens
  $g_{i,t} \leftarrow \mathrm{sg}[\sigma(\beta(\ell^T_{i,t} - \mathrm{sg}[\ell^S_{i,t}(\theta)]))]$
  $\mathcal{L}_{\mathrm{gate}} \leftarrow \mathrm{Agg}_{\mathcal{P}}[g_{i,t}(\ell^T_{i,t} - \ell^S_{i,t}(\theta))]$
  $\mathcal{L}_{\mathrm{plain}} \leftarrow \mathcal{L}^{\mathcal{P}}_{\mathrm{PG}}(\theta; A) + \alpha\mathcal{L}_{\mathrm{gate}}$
  Form $\mathcal{L}_{\mathrm{replay}} \leftarrow \mathcal{L}^{\mathcal{R}}_{\mathrm{PG}}(\theta; A^{\mathcal{R}})$ if replays are present
  Update θ to minimize $\mathrm{Combine}(\mathcal{L}_{\mathrm{plain}}, \mathcal{L}_{\mathrm{replay}})$
end for
return θ

Algorithm 5 StudentUpdate with RLSD-style advantage reweighting

Input: θ, plain batch $\mathcal{P}$, replay batch $\mathcal{R}$, step s, λ_0, W, ϵ_w
Output: Updated student parameters θ

Compute reward-derived advantages for $\mathcal{P}$ and, if present, $\mathcal{R}$
$\lambda_s \leftarrow \lambda_0 \max(0, 1 - s/W)$
for plain response token (i, t) **do**
  $\delta_{i,t} \leftarrow \ell^T_{i,t} - \ell^{\mathrm{old}}_{i,t}$
  $w_{i,t} \leftarrow \mathrm{clip}(\exp(\mathrm{sign}(A_i)\delta_{i,t}), 1 - \epsilon_w, 1 + \epsilon_w)$
  $\widetilde{A}_{i,t} \leftarrow A_i[(1 - \lambda_s) + \lambda_s w_{i,t}]$
end for
for optimization minibatch **do**
  $\mathcal{L}_{\mathrm{plain}} \leftarrow \mathcal{L}^{\mathcal{P}}_{\mathrm{PG}}(\theta; \widetilde{A})$
  Form $\mathcal{L}_{\mathrm{replay}} \leftarrow \mathcal{L}^{\mathcal{R}}_{\mathrm{PG}}(\theta; A^{\mathcal{R}})$ if replays are present
  Update θ to minimize $\mathrm{Combine}(\mathcal{L}_{\mathrm{plain}}, \mathcal{L}_{\mathrm{replay}})$
end for
return θ

In both algorithms, $\mathcal{L}_{\mathrm{PG}}$ uses the current-to-sampling-policy ratios of the policy being updated. The aggregation of replay and plain trajectories follows their valid rows and tokens, rather than an imposed equal mixture.

A.5 Baseline Training Algorithms

Algorithms 6–9 summarize the baseline update rules using the same rollout notation, following the algorithm-by-algorithm presentation of Lu et al. (2026). They describe supervision and optimization, not additional reproductions of the literature results. Baseline-specific teacher construction, distillation estimators, schedules, and regularization follow their reported protocols (Section 4.1). SDAR and RLSD use Algorithms 4 and 5 with their baseline teacher context and no EvoHint replay batch.

Let $\mathcal{L}_D(\theta; \mathcal{P}, T)$ denote the baseline's on-policy distillation objective for teacher T. Teacher outputs are detached. The teacher may be a fixed reference or a privileged-context self-teacher, as specified by the baseline; it is not the trainable hint generator G_ϕ.

Algorithm 6 GRPO: outcome-based policy optimization

Input: Student π_θ, tasks $\mathcal{D}$, rollouts per task N, clipping configuration
Output: Trained student π_θ

for training iteration **do**
  Snapshot $\theta_{\text{old}} \leftarrow \theta$; sample a task batch
  Generate N attempts per task with $\pi_{\theta_{\text{old}}}$; evaluate task rewards
  Compute within-task reward advantages; broadcast them to valid response tokens
  Cache sampling-time log probabilities and response masks
  for optimization minibatch **do**
    Compute current-to-sampling-policy probability ratios on the sampled tokens
    Update θ with the clipped loss $\mathcal{L}_{\text{PG}}(\theta; A)$
  end for
end for
return π_θ

Algorithm 7 OPSD: on-policy distillation without an outcome-policy loss

Input: Student π_θ, tasks $\mathcal{D}$, rollouts per task N, teacher-context procedure, distillation coefficient α
Output: Trained student π_θ

for training iteration **do**
  Snapshot $\theta_{\text{old}} \leftarrow \theta$; sample a task batch
  Collect $\mathcal{P}$ with N plain attempts per task from $\pi_{\theta_{\text{old}}}$
  Construct the prescribed teacher T and obtain detached outputs on student prefixes
  for optimization minibatch **do**
    Compute $\mathcal{L}_D(\theta; \mathcal{P}, T)$ with the prescribed distillation estimator
    Update θ to minimize $\alpha\mathcal{L}_D$
  end for
end for
return π_θ

Algorithm 8 Skill-SD: skill-conditioned distillation with outcome-based RL

Input: Student π_θ, tasks $\mathcal{D}$, skill library, rollouts per task N, distillation schedule λ_s
Output: Trained student π_θ

for training iteration s **do**
  Snapshot $\theta_{\text{old}} \leftarrow \theta$; sample tasks and retrieve task-relevant skills
  Collect N plain attempts per task, rewards, old log probabilities, and advantages A
  Cache detached skill-conditioned self-teacher scores $\ell^T_{i,t}$
  for optimization minibatch **do**
    Recompute student scores $\ell^S_{i,t}(\theta)$
    $d_{i,t} \leftarrow \ell^S_{i,t}(\theta) - \ell^T_{i,t}$
    $k_{i,t} \leftarrow \exp(-d_{i,t}) - 1 + d_{i,t}$
    $\rho_{i,t} \leftarrow \exp(\ell^S_{i,t}(\theta) - \ell^{\text{old}}_{i,t})$
    $\mathcal{L}_D \leftarrow \text{Agg}[\rho_{i,t} k_{i,t}]$
    Update θ with $\mathcal{L}_{\text{PG}}(\theta; A) + \lambda_s \mathcal{L}_D$
  end for
end for
return π_θ

Algorithm 8 displays the importance-weighted K_3 estimator used by the archived Skill-SD implementation. The token ratio corrects the sampled action distribution at each fixed prefix; it is not a full-trajectory importance ratio. Its coefficient schedule and numerical stabilization follow the baseline configuration; it has no learned hint generator or outcome-selected EvoHint memory updates.

Algorithm 9 GRPO+OPSD: additive outcome and distillation objectives

Input: Student π_θ, tasks $\mathcal{D}$, rollouts per task N, teacher-context procedure, coefficient α
Output: Trained student π_θ

for training iteration **do**
  Snapshot $\theta_{\text{old}} \leftarrow \theta$; sample a task batch
  Collect $\mathcal{P}$ with N plain attempts per task and evaluate rewards
  Compute group-relative advantages A and cache old log probabilities
  Construct teacher T; cache detached outputs on the same student prefixes
  for optimization minibatch **do**
    Compute $\mathcal{L}_{\text{PG}}(\theta; A)$ and $\mathcal{L}_D(\theta; \mathcal{P}, T)$
    Update θ to minimize $\mathcal{L}_{\text{PG}} + \alpha \mathcal{L}_D$
  end for
end for
return π_θ

Skill-GRPO uses Algorithm 6 with retrieved skills in the rollout context. Skill-Prompt adds retrieved skills only at inference; Vanilla and Base perform no task-specific updates. The imported math GRPO+SDPO baseline also combines outcome and distillation objectives, with the solution-conditioned teacher and KL estimator reported by Li et al. (2026b); its numerical implementation should not be inferred from the agent OPSD estimator.