REVES: REvision and VErification–Augmented Training for Test-Time Scaling

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Test-time scaling via sequential revision has emerged as a powerful paradigm for enhancing Large Language Model (LLM) reasoning. However, standard post-training methods primarily optimize single-shot objectives, creating a fundamental misalignment with multi-step inference dynamics. While recent work treats this as multi-turn reinforcement learning (RL), conventional approaches optimize over the multi-step trajectories directly, failing to further exploit the high-quality mistakes in intermediate steps that model can learn from correcting them. We propose a two-stage iterative framework that alternates between online data/prompt augmentation and policy optimization. By converting the intermediate steps ("near-miss" answers) in the successful recovery trajectories into decoupled revision and verification prompts, our approach concentrates training on both effective answer transformation and error identification. This approach enables efficient off-policy data generation and reduces the computational overhead of long-horizon sampling compared to standard multi-turn RL. On LiveCodeBench, using publicly available test cases as feedback, we observe gains of +6.5 points over the RL baseline and +4.0 points over standard multi-turn training. Beyond coding, our approach matches the previously reported SOTA result on circle packing. Math results under ground-truth verification further confirm improved correction ability. It also generalizes to out-of-distribution constraint-satisfaction puzzles such as n_queens and mini_sudoku, where correctness is defined entirely by problem constraints.

1 Introduction

Large language models (LLMs) deployed in challenging settings rarely produce a correct answer on the first attempt; real-world workflows routinely loop back with feedback. A growing family of test-time scaling (TTS) algorithms formalizes this pattern, including sequential revision (Madaan et al., 2023; Shinn et al., 2023), tree search (Inoue et al., 2025), evolutionary refinement (Lee et al., 2025b), and best-of-N with a verifier (Chow et al., 2025). They all share a single primitive: given a problem, a previous attempt, and feedback, produce an improved response.

This raises a basic question: *can we design a post-training algorithm that explicitly improves a model's test-time sequential revision capability?* We focus on sequential revision (SR) because gains on SR transfer to the rest of the family: every revision-using TTS algorithm eventually calls the policy on revise-from-prior-attempt inputs, and SR has the broadest such distribution. Theorem 3.1 shows that increasing the policy's one-step revision value on SR-induced inputs also raises J_ϕ for every revision-using TTS algorithm whose call inputs are covered by SR, up to small drift terms.

How, then, do we post-train an LLM to optimize the SR objective $J_{\phi_{\mathrm{SR}}}$? Standard post-training, RLHF (Ouyang et al., 2022), RLVR, and GRPO (DeepSeek-AI et al., 2025), optimizes a single-shot expected reward, which is fundamentally misaligned with the multi-step nature of test-time

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

deployment. The natural fix is multi-turn RL on revision rollouts, but trajectory-level credit broadcasts assign path-dependent credit. Take a wrong, wrong, correct rollout: every per-turn log-probability gradient receives the same positive credit, including the wrong intermediates. Per rollout this credit is biased, and only cancels in expectation across many rollouts.

We propose REVES, a two-stage iterative framework that targets the SR objective $J_{\phi_{\mathrm{SR}}}$ directly through its per-state structure. We show that $J_{\phi_{\mathrm{SR}}}$ decomposes exactly into a weighted sum of per-state one-step recovery probabilities along visited rollouts (Lemma 4.1), exposing single-state, single-turn gradients with no horizon credit assignment. REVES is an offline realization of this per-state training signal: each epoch, it runs SR rollouts under the current policy, retains those that succeed within budget, converts the intermediate ("near-miss") answers into decoupled *revision* and *verification* prompts, and trains with standard single-turn RL on the augmented prompt set; the next epoch refreshes the augmentation. This concentrates training on per-state recovery and avoids both the bias of trajectory broadcasts and the cost of long-horizon online sampling.

Empirically, on LiveCodeBench with public test cases as feedback, REVES achieves $+6.5$ points over the single-shot RL baseline and $+4.0$ points over standard multi-turn training. On the circle packing benchmark, a Qwen3-4B base trained with REVES matches the previously reported best results from much larger evolutionary search systems built on Gemini-2.0 Pro/Flash and Qwen3-8B. Math benchmarks (MATH500, AIME24/25) show consistent improvements under both ground-truth and self-confidence stopping, and the trained policies generalize to out-of-distribution constraint-satisfaction puzzles (n_queens, mini_sudoku). Beyond standalone SR, REVES improves every revision-using TTS algorithm we tested (BoN, MCTS, AB-MCTS variants, Mind Evolution), in line with the conditional bound in Theorem 3.1.

Our contributions are threefold:

1. **Conceptual.** We formulate test-time scaling as a meta-RL problem and show that improving sequential revision also improves any revision-using TTS algorithm whose call inputs are covered by SR (Theorem 3.1).
2. **Methodological.** We derive an exact decomposition of $J_{\phi_{\mathrm{SR}}}$ into per-state one-step recovery probabilities and propose REVES (Lemma 4.1), a two-stage framework that converts intermediate near-miss answers into decoupled revision and verification prompts and trains with single-turn RL, sidestepping the path-dependent credit assignment of multi-turn RL.
3. **Empirical.** REVES delivers consistent test-time improvements across math, coding, out-of-distribution puzzles, circle packing, and improves revision-using TTS algorithm we evaluated.

2 Related Work

Test-time-aware post-training. A growing line of work shows the importance of aligning the post-training objective with the intended test-time inference strategy. For parallel-sampling families, this includes optimizing pass@k (Chen et al., 2025; Walder and Karkhanis, 2025; Tang et al., 2025) when deployment uses best-of-N, majority voting (Wang et al., 2023), or learned verifiers (Chow et al., 2025). Our deployment is structurally different: the test-time strategies we care about, including sequential revision, MCTS variants, and Mind Evolution, are all *revision-using* algorithms whose calls take as input a previous attempt together with its feedback. Aligning training with this inference family therefore requires a different objective.

Training sequential revision capability. Closer to our setting, several prior methods explicitly train the policy to revise, including supervised and preference-based approaches (Qu et al., 2024; Xiong et al., 2025), RL-based approaches (Kumar et al., 2024; Jiang et al., 2025; Lee et al., 2025a), and multi-turn RL methods that incorporate per-turn natural-language or numerical critiques (Jain et al., 2025; Li et al., 2025b,a; Zhang et al., 2026). Our work shares the broader goal of strengthening the model's revision capability, but differs in framing: we primarily focus on how to align training with the test-time objective of a revision-using inference strategy.

Weakness-driven data synthesis and guided exploration. SwS (Liang et al., 2025) synthesizes new problems targeting the model's weaknesses using strong external teacher models, and POPE (Qu et al., 2026) uses privileged oracle hints to guide on-policy exploration on hard problems. These methods are orthogonal to ours: they target single-shot pass@1, whereas REVES targets a fundamentally different test-time objective.

3 Test-Time Scaling as a Meta-RL Problem

Setup and two motivating examples. We fix notation used throughout the paper. Denote by $\mathcal{X}$ the distribution over input problems, $\mathcal{Y}$ the answer space, and $r^\star : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ the *ground-truth* reward, where $r^\star(x, y) = 1$ indicating a fully correct answer. We focus on the verifiable-reward regime where $r^\star$ is computable during both training and at test time, e.g., public test-case execution for code, and constraint satisfaction for puzzles. When $r^\star$ is unavailable at test time, a surrogate stopping rule replaces it (Section 4). We model the LLM as a policy π_θ in RL parameterized by θ. At deployment, an LLM rarely operates by a single call to π_θ; instead a *test-time scaling (TTS) algorithm* ϕ orchestrates up to K policy calls and extracts a final answer $\hat{y} \in \mathcal{Y}$. Two canonical examples bracket the design space: *Best-of-N (BoN)* (Chow et al., 2025) draws N candidates in parallel from $\pi_\theta(\cdot \mid x)$ and selects $\hat{y}$ via an external scorer; every call uses the same original prompt. *Sequential Revision (SR)* (Madaan et al., 2023) generates an initial response $y_1 \sim \pi_\theta(\cdot \mid x)$ from prompt x, and for $t = 2, \ldots, K$ produces a revision $y_t \sim \pi_\theta(\cdot \mid x, y_{t-1}, f_{t-1})$ conditioned on the previous attempt y_{t-1} and a feedback signal f_{t-1} (e.g., compiler error messages in code generation), halting at the first correct response. Detailed pseudocode is in Protocol 1 of Appendix A.

General formulation: the TTS-induced decision process. Pairing any TTS algorithm ϕ with π_θ induces a decision process with prompt-state space $\mathcal{S}$ and response space $\mathcal{Y}$. At each step t, the policy emits a response $y_t \sim \pi_\theta(\cdot \mid s_t)$, and ϕ constructs the next prompt via a transition kernel $P_\phi(s_{t+1} \mid s_{1:t}, y_{1:t})$. The initial state is $s_1 = x$. Note that the policy itself remains Markov in the current state (at every step, π_θ conditions only on the local prompt s_t); the history dependence is in the kernel, because ϕ may branch from any earlier response. We focus on the family $\Phi_{\mathcal{R}}$ of TTS algorithms that issue at least one revision call on a prior attempt; this includes Sequential Revision and adaptive tree search such as AB-MCTS (Inoue et al., 2025). For such algorithms a revision-step prompt has the form $s_{t+1} = (x, y_{i_t}, f_{i_t}),\ i_t \in \{1, \ldots, t\}$, where the prior step i_t is chosen by ϕ: for SR, $i_t = t$ (always the most recent response), so s_{t+1} depends only on (s_t, y_t) and the induced process is a standard MDP; for tree-search algorithms such as AB-MCTS, i_t may be any earlier response that ϕ's search policy decides to branch from, and the kernel is genuinely history-dependent. The procedure runs until a random stopping time $\tau := K \wedge \min\{t : r^\star(x, y_t) = 1\}$ and outputs $\hat{y} = y_\tau$; that is, the procedure either *early-stops* once a verified-correct response is produced, or *exhausts the budget* K. Running π_θ inside ϕ yields the TTS-induced objective

$$J_\phi(\theta) = \mathbb{E}_{x \sim \mathcal{X},\, (s_{1:\tau}, y_{1:\tau}) \sim (\pi_\theta, \phi)} \big[r^\star(x, y_\tau) \big],$$

to be contrasted with the standard single-shot objective

$$J_{\text{OneShot}}(\theta) = \mathbb{E}_{x \sim \mathcal{X},\, y \sim \pi_\theta(\cdot \mid x)} \big[r^\star(x, y) \big],$$

which is what RLHF (Ouyang et al., 2022), RLVR, and GRPO (DeepSeek-AI et al., 2025) optimize. Optimizing J_ϕ rather than J_{OneShot} is a *meta-RL* problem (Figure 1): the policy must adapt across the distribution of multi-step contexts that ϕ generates, not just answer the original question. Our goal is to improve J_ϕ for every TTS algorithm $\phi \in \Phi_{\mathcal{R}}$.

This raises an important question: *does there exist a training target $J_{\phi_{\text{train}}}$ whose optimization provably improves J_ϕ for every $\phi \in \Phi_{\mathcal{R}}$?* If so, we can train against one algorithm and have the gains transfer to every other deployment-time choice in $\Phi_{\mathcal{R}}$, side-stepping the cost of running tree search or external verifiers during training. The answer is affirmative. The key fact is that every algorithm in $\Phi_{\mathcal{R}}$ at some point invokes π_θ to revise a previous attempt, so a training target whose own revision-input distribution covers the deployment one improves every member of $\Phi_{\mathcal{R}}$ when its revision-value goes up.

Theorem 3.1 (Sequential-revision recovery transfers to revision-using TTS)**.** *Let π_0 be a baseline policy and π_1 an updated policy. For $\phi \in \Phi_{\mathcal{R}}$, policy π, and revision-call input z, let*

$$V_\pi(z) := \mathbb{P}_{y \sim \pi(\cdot \mid z)} \big(r^\star(x, y) = 1 \big), \qquad \rho_\pi^\phi(z) := \mathbb{E}_{(\phi, \pi)} \Big[\textstyle\sum_{t=1}^{\tau} \mathbf{1}\{z_t = z\} \Big]$$

denote the one-step recovery probability and the expected visit count to z. Assume:

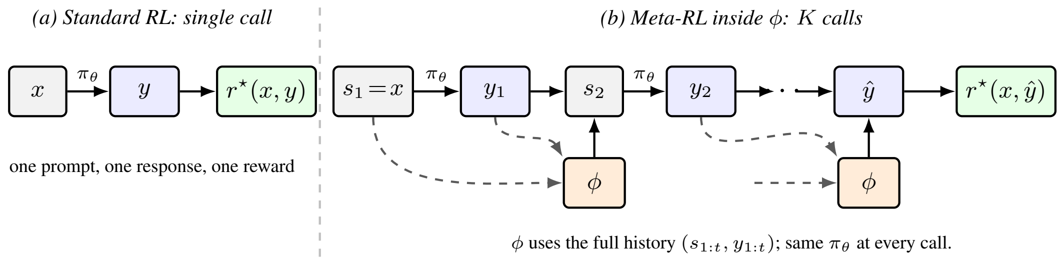


Figure 1: **(a)** Standard RL: policy π_θ maps problem x to response y with reward $r^\star(x, y)$. **(b)** A TTS algorithm ϕ invokes the same π_θ over up to K steps, each prompt s_t built from the prior history (dashed arrows), and extracts a final response $\hat{y} = y_\tau$ at stopping time $\tau \leq K$; π_θ must therefore behave well across the whole context distribution induced by ϕ, not just on x.

(C1) Coverage by SR at π_0: *there exists* $C_\phi \geq 1$ *such that* $\rho^{\phi}_{\pi_0}(z) \geq \frac{1}{C_\phi}\, \rho^{\phi_{\text{SR}}}_{\pi_0}(z)$ *for every* z.

(C2) Monotone recovery improvement: $V_{\pi_1}(z) \geq V_{\pi_0}(z)$ *for every* $z \in \operatorname{supp}(\rho^{\phi}_{\pi_0})$.

Then, we have the following guarantee:

$$J_\phi(\pi_1) - J_\phi(\pi_0) \;\geq\; \frac{1}{C_\phi}\big[J_{\phi_{\text{SR}}}(\pi_1) - J_{\phi_{\text{SR}}}(\pi_0)\big] \;-\; \big\|\rho^{\phi}_{\pi_1} - \rho^{\phi}_{\pi_0}\big\|_1 \;-\; \frac{1}{C_\phi}\big\|\rho^{\phi_{\text{SR}}}_{\pi_1} - \rho^{\phi_{\text{SR}}}_{\pi_0}\big\|_1.$$

The take-away is clear: improving the SR test-time objective $J_{\phi_{\text{SR}}}$ also improves J_ϕ for every revision-using $\phi \in \Phi_{\mathcal{R}}$, so we focus on SR as the training target. See Appendix C.2 for the proof.

Our training target $J_{\phi_{\text{SR}}}$. The theorem licenses any $\phi_{\text{train}} \in \Phi_{\mathcal{R}}$ whose visit measure dominates the deployment-time ϕ's. We pick ϕ_{SR} and optimize

$$J_{\phi_{\text{SR}}}(\theta) := \mathbb{E}_{x \sim \mathcal{X},\, y_{1:\tau} \sim (\pi_\theta, \phi_{\text{SR}})}\big[r^\star(x, y_\tau)\big]$$

for two reasons. First, ϕ_{SR} has the broadest revision-input distribution in $\Phi_{\mathcal{R}}$. At every step, the next prompt is conditioned on a freshly drawn y_{t-1}, so successive revision inputs z_t are spread out. Tree-search and selection algorithms instead branch repeatedly from a small pool of cached responses, concentrating their visits on a narrow subset. (C1) therefore holds at the smallest C_ϕ over deployment-time ϕ. Second, ϕ_{SR} is itself a strong TTS algorithm. Figure 2 shows that on TravelPlanner (Xie et al., 2024), ϕ_{SR} matches or surpasses Mind Evolution (Lee et al., 2025b) and AB-MCTS (Inoue et al., 2025) (left), with the Markov-revision form $\pi_\theta(\cdot \mid x, y_{\text{prev}}, f)$ on the best cost-accuracy frontier (right), while being the simplest to implement.

Optimizing $J_{\phi_{\text{SR}}}$ is not optimizing single-shot pass@1. A natural next question is whether standard single-shot RLVR optimization already addresses $J_{\phi_{\text{SR}}}$. It does not: two policies can be *indistinguishable under* J_{OneShot} on every problem in $\mathcal{X}$ yet differ substantially under $J_{\phi_{\text{SR}}}$.

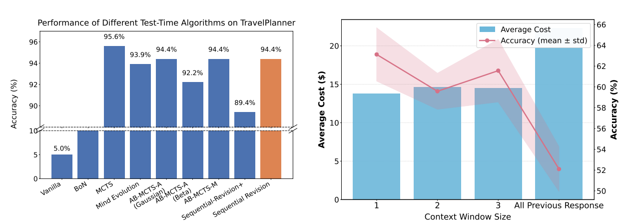


Figure 2: DeepSeek-V3 on TravelPlanner. **Left:** sequential revision vs. Mind Evolution / MCTS / AB-MCTS. **Right:** cost-accuracy as the revision context window covers the k most recent responses, $k \in \{1, 2, 3\}$, or the full history.

Theorem 3.2 (Objective mismatch)**.** *For any revision length* $K \geq 2$, *there exist policies* π_1, π_2, $\Delta(K) > 0$, *and a problem distribution* $\mathcal{X}$ *such that* $J_{\text{OneShot}}(\pi_1) = J_{\text{OneShot}}(\pi_2)$, *while* $J_{\phi_{\text{SR}}}(\pi_2) - J_{\phi_{\text{SR}}}(\pi_1) \geq \Delta(K)$.

The implication is direct: J_{OneShot} and $J_{\phi_{\text{SR}}}$ are different objectives, and policies with identical single-shot performance can have substantially different sequential-revision performance. Our problem reduces to a single concrete question: how do we post-train π_θ so that $J_{\phi_{\text{SR}}}$ goes up? The next section presents our algorithm.

4 REVES: From Path-Level Credit to Recovery-Level Supervision

Section 3 established that improving $J_{\phi_{\mathrm{SR}}}$ is our focus. This section turns that target into an algorithm. We start by writing $J_{\phi_{\mathrm{SR}}}$ exactly as a sum of per-state one-step recovery probabilities along visited trajectories; this hazard decomposition makes the natural training signal explicit and reads off the algorithm directly from the figure.

Lemma 4.1. *Let $\zeta = (z_1, y_1, \dots, z_\tau, y_\tau) \sim (\pi_\theta, \phi_{\mathrm{SR}})$ be a stopped SR trajectory under π_θ: at each step t the model samples $y_t \sim \pi_\theta(\cdot \mid z_t)$ and stops at the first verified correct response or at budget K, with $z_t = (x, y_{t-1}, f_{t-1})$ the revision state before generating y_t. Let $V_\pi(z_t) := \mathbb{P}_{y' \sim \pi_\theta(\cdot|z_t)}\big(r^\star(x, y') = 1\big)$ denote the one-step recovery probability at z_t, where y' is a fresh sample (not the trajectory's own y_t), and let $\rho_\theta(z) := \mathbb{E}_{x\sim\mathcal{X},\, \zeta\sim(\pi_\theta,\phi_{\mathrm{SR}})}\Big[\sum_{t=1}^{\tau} \mathbf{1}\{z_t = z\}\Big]$ denote the expected number of times state z is visited along a SR rollout. Then*

$$J_{\phi_{\mathrm{SR}}}(\theta) = \sum_{t=1}^{K} \mathbb{E}_{x,\zeta}\big[\mathbf{1}\{\tau \geq t\}\, V_\pi(z_t)\big] = \mathbb{E}_{x\sim\mathcal{X},\, \zeta\sim(\pi_\theta,\phi_{\mathrm{SR}})}\left[\sum_{t=1}^{\tau} V_\pi(z_t)\right]$$
$$= \sum_{z} \rho_\theta(z)\, \mathbb{E}_{y'\sim\pi_\theta(\cdot|z)}\big[r^\star(x, y')\big]. \tag{1}$$

See Appendix C.3 for the proof. Equation (1) reads the algorithm off Figure 3 (b). The horizontal SR trajectory provides the outer samples $z \sim \rho_\theta$, and at each visited state $z = (x, y_{t-1}, f_{t-1})$ the branches drawn from $\pi_\theta(\cdot \mid z)$ are online rollouts that estimate and optimize the per-state factor $V_\pi(z) = \mathbb{E}_{y'\sim\pi_\theta(\cdot|z)}[r^\star(x, y')]$. Since $J_{\phi_{\mathrm{SR}}}$ is a nonnegative-weighted sum of these factors, raising V_π at any visited z directly raises $J_{\phi_{\mathrm{SR}}}$, without horizon credit assignment.

Comparison with multi-turn RL. Multi-turn RL provides an unbiased policy-gradient estimator for the trajectory-level objective. However, its per-turn learning signal is highly indirect. Consider a wrong–wrong–correct rollout: the gradients for all three turns are weighted by the same positive terminal advantage, including the two turns that produced incorrect intermediate responses. This is valid for optimizing terminal success of the whole trajectory, but it yields a high-variance and coarse signal for improving the local revision behavior at each intermediate state.

Our method instead reuses such incorrect intermediate responses as revision states. Given a state $z = (x, y_{\mathrm{prev}}, f_{\mathrm{prev}})$, we draw fresh samples from $\pi(\cdot \mid z)$ and estimate the local one-step recovery probability $V_\pi(z) = \Pr_{y\sim\pi(\cdot|z)}\big[r^\star(x, y) = 1\big]$. Thus, the intermediate wrong response is not treated as an action to imitate; it is treated as a state from which the model should learn to recover. This converts a successful long-horizon trajectory into several short-horizon revision problems, providing a sharper training signal for local recovery while relying on the current policy to discover useful intermediate states.

From recovery target to algorithm. An ideal on-policy implementation of Lemma 4.1 draws $z \sim \rho_\theta$ by running SR and runs single-turn RL at each z. Three design choices turn this into the algorithm we use.

(1) Verification prompts alongside revision data. At test time the verifier is typically unavailable. We therefore include both *revision prompts* and *verification prompts* (asking the policy to judge whether an intermediate response is correct) in the augmented data, so the trained policy can self-stop at deployment. The split of gains across the two prompt types is in Table 4.

(2) Offline generation. An on-policy estimator of $\nabla J_{\phi_{\mathrm{SR}}}$ would need fresh SR rollouts every gradient step, which is intrinsically serial and prohibitively slow. We instead generate visited states once per epoch in Stage I, reuse them through Stage II, and refresh at the next epoch.

(3) Successful-trajectory filtering. For a far-from-optimal base policy, online samples at states from failed rollouts rarely produce a verifier-accepted recovery and waste tokens. We restrict Stage I to states on trajectories that succeed within budget K, where at least one recovery target exists by construction; the filter loosens as π_θ improves.

Two-stage realization. **Stage I (Data augmentation).** For each input x, run SR under the current π_θ: $y_1 \sim \pi_\theta(\cdot \mid x)$ and $y_t \sim \pi_\theta(\cdot \mid x, y_{t-1}, \texttt{Feedback}(y_{t-1}))$ for $t = 2, \dots, K$, terminating at

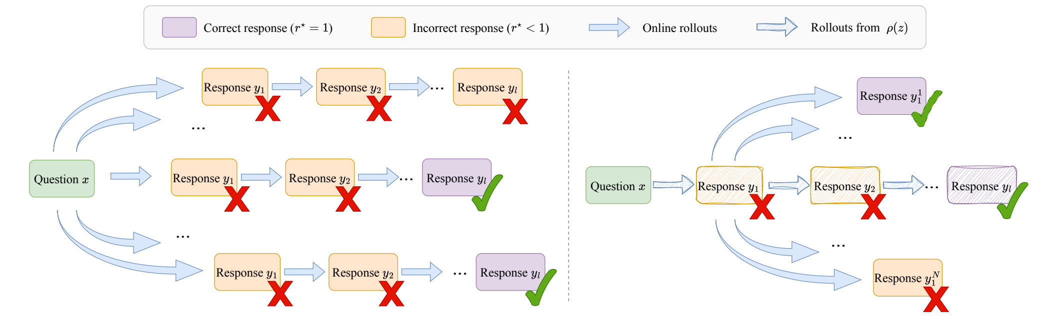


Figure 3: Mapping Eq. (1) onto the figure. **(a)** Stage I samples SR rollouts under the current π_θ and keeps those that succeed within budget K. **(b)** The horizontal hatched chain is a retained sequential-revision trajectory; it determines the visit-count weight $\rho_\theta(z)$ at each intermediate state $z = (x, y_{t-1}, f_{t-1})$. The branches at each visited z are fresh samples from $\pi_\theta(\cdot \mid z)$; their verifier outcomes provide the per-state estimate of $V_\pi(z) = \mathbb{E}_{y' \sim \pi_\theta(\cdot|z)}[r^\star(x, y')]$.

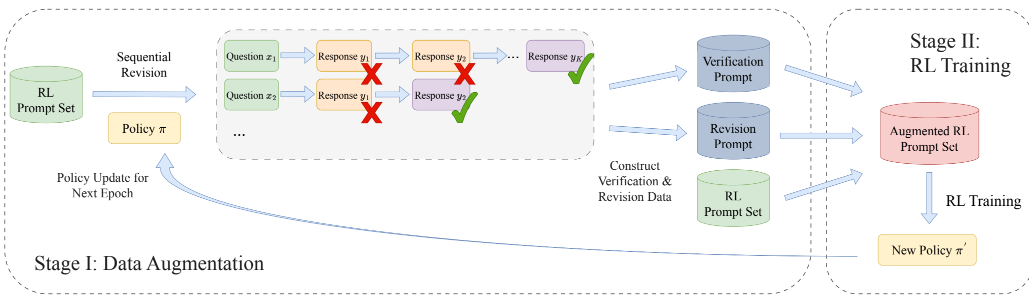


Figure 4: Overview of the proposed RL-based data augmentation framework. In Stage I, the current policy performs sequential revision; only trajectories that succeed within budget K are retained, and their intermediate states yield revision and verification prompts that are merged with the original RL prompt set. In Stage II, the policy is updated by single-turn RL on the augmented set, and the new policy regenerates the augmentation in the next epoch.

the first correct response or after K attempts. Discard trajectories that fail or are already correct at y_1. From each retained trajectory $(x, y_1, \ldots, y_\ell)$ with $r^\star(x, y_\ell) = 1$, build (i) *revision data* $\{\texttt{Revision Prompt}(x, y_i)\}_{i=1}^{\ell}$, where each prompt asks the model to revise the previous response, and (ii) *verification data* $\{\texttt{Verification Prompt}(x, y_i)\}_{i=1}^{\ell}$, where each prompt asks the model to judge whether y_i is correct. Templates are in Appendix A. **Stage II (Single-turn RL).** Train π_θ on the union of the original RL prompts and the prompts produced in Stage I. The updated policy serves as the rollout policy for the next epoch's Stage I. Figure 4 shows the loop.

Efficiency. Our algorithm is also more efficient than standard multi-turn training. First, augmented samples are generated in an *off-policy* manner. With a well-designed infrastructure, data augmentation can run in parallel with RL training, so the model continues training on the original RL prompts while new augmented data are generated asynchronously, avoiding any interruption to the main training loop. Second, training itself is single-turn. Each gradient step processes one (prompt, response) pair as in ordinary RLVR, with no per-step sequential rollout in the inner loop, which is the dominant cost of multi-turn RL with budget K. Appendix F.2 reports the wall-clock comparison.

Test-time stopping rule. At test time the oracle stopping time τ is almost always unavailable, so we adopt a surrogate stopping time $\hat{\tau}$ that replaces $r^\star$ with a domain-specific signal.

For mathematical reasoning tasks, we leverage the *Tail Confidence* metric of Fu et al. (2025) as the surrogate signal. At each revision step t, the token-level confidence at position i of y_t is $C_i = -\frac{1}{k}\sum_{j=1}^{k} \log P_i(j)$, where $P_i(j)$ is the probability of the j-th most likely token (a larger C_i indicates a more concentrated, more confident distribution). The response-level score c_t averages C_i over the tail positions $i \in T_{\text{tail}}$ of y_t. The stopping time is $\hat{\tau} := K \wedge \min\Big\{t \geq 3 : c_t / \sum_{j=1}^{t} c_j > c\Big\}$

for a predefined threshold c. If $\hat{\tau} = K$ (budget exhausted without trigger), we output $\hat{y} = y_{t^*}$ with $t^* = \arg\max_{1 \le t \le K} c_t$, breaking ties uniformly at random when several responses share the same confidence. Selecting the most confident response across all K revisions instead of just y_K improves robustness under small budgets by mitigating noise in confidence estimates at the stopping step.

For coding tasks, we use publicly available test cases as a surrogate reward $\tilde{r} : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$, with $\tilde{r}(x, y) = \mathbf{1}(y \text{ passes all public test cases})$. The stopping time is $\hat{\tau} := K \wedge \min\{t : \tilde{r}(x, y_t) = 1\}$, and we output $\hat{y} = y_K$ when $\hat{\tau} = K$.

5 Experimental Results

Models and data. We train Qwen2.5-7B, Qwen2.5-3B (Yang et al., 2024), Qwen3-4B(non-thinking) (Yang et al., 2025) and DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI et al., 2025) directly with RL. Training data is sampled from `Skywork/Skywork-OR1-RL-Data` (7298 math + 1015 coding = 8313 examples). We evaluate on MATH500 and AIME 24/25 for math; LiveCodeBench (Jain et al., 2024) and CodeContest (Li et al., 2022) for coding; and n_queens / mini_sudoku from ReasoningGym (Stojanovski et al., 2025) for out-of-distribution puzzles.

Baselines and implementation. We compare against **RL** (single-response RL, no revision), **Multi-Turn** (multi-turn rollouts without explicit self-verification), and **PAG** (Jiang et al., 2025) (multi-turn with self-verification via a designed reward). Full details in Appendix E.

5.1 Main Results

Consistent improvements on coding tasks. Table 1 demonstrates the benefits of RL-Augmentation on coding benchmarks. Under test-time surrogate stopping criteria, where publicly available test cases are used to determine when to stop and execution results are provided as feedback for subsequent revisions (see Appendix A for an example), RL-Augmentation consistently improves performance across different test-time budgets, demonstrating robustness beyond symbolic rewards. Additional results on DeepSeek-R1-Distill-7B (DeepSeek-AI et al., 2025) are provided in Appendix G.

Table 1: Sequential-revision test-time scaling performance on coding benchmarks (LiveCodeBench, CodeContest). The best result in each column is shown in bold, and the second-best is underlined. TestCases-B is sequential-revision (SR) test-time scaling with budget B, using execution-based test cases as the verifier and feeding the resulting feedback into the next revision step. Best results in each column are in bold; second-best are underlined.

Model	LiveCodeBench (08/01/24–01/01/25)			LiveCodeBench (01/01/25–05/01/25)		CodeContest		
	OneShot	TestCases-32	TestCases-4	OneShot	TestCases-32	OneShot	TestCases-32	TestCases-4
Qwen-2.5-7B	4.78	6.13	6.13	6.04	18.86	0.41	5.73	5.86
Qwen-2.5-7B-RL	18.02	23.04	20.83	19.96	24.18	**7.07**	11.09	7.47
Qwen-2.5-7B-Multi-turn	19.36	25.49	21.33	**20.70**	27.47	6.87	14.34	8.08
Qwen-2.5-7B-PAG	**19.49**	25.73	23.16	19.78	26.37	6.46	14.95	9.09
Qwen-2.5-7B-RL-Aug	18.87	**29.54**	**23.77**	20.51	**30.03**	**7.07**	**16.57**	**11.72**
Qwen-2.5-3B	5.88	15.81	7.23	7.87	18.31	1.62	6.26	2.43
Qwen-2.5-3B-RL	9.19	12.01	11.52	14.11	17.76	3.03	4.24	3.64
Qwen-2.5-3B-Multi-turn	**9.93**	15.69	**12.13**	13.19	19.23	3.23	7.88	4.44
Qwen-2.5-3B-PAG	8.09	14.71	11.15	13.37	18.50	**3.64**	5.66	4.85
Qwen-2.5-3B-RL-Aug	9.56	**17.65**	11.03	**14.29**	**21.61**	2.22	**8.48**	**5.45**
Qwen3-4B (non-thinking)	17.60	22.10	20.20	20.90	22.00	8.30	12.80	7.70
Qwen3-4B-RL	22.40	30.90	29.00	**26.90**	29.70	**21.80**	28.20	26.30
Qwen3-4B-Multi-turn	20.20	30.10	28.30	24.70	31.90	11.50	**39.10**	28.80
Qwen3-4B-PAG	22.40	27.90	24.30	23.10	28.60	10.90	32.10	28.20
Qwen3-4B-RL-Aug	**23.90**	**32.40**	**30.90**	25.80	**33.50**	14.10	37.20	**32.70**

Improved sequential revision capability for mathematical reasoning. Table 2 shows that RL-Augmentation substantially improves sequential revision performance across all evaluated math benchmarks. Under oracle stopping (the Oracle-32 and Oracle-4 columns, where the rollout halts as soon as a generated answer matches the ground truth, isolating the model's ability to revise an incorrect solution into a correct one), our method achieves consistently large gains. These improvements persist under the SelfConf-4 column, the practical test-time stopping rule based on the model's own confidence, indicating that the learned capability transfers beyond the oracle setting.

Method	Model	$n = 26$ (↑)
AlphaEvolve (Novikov et al., 2025)	Gemini-2.0 Pro + Flash	2.635862
AlphaEvolve V2 (Georgiev et al., 2025)	Gemini-2.0 Pro + Flash	2.635983
ShinkaEvolve (Lange et al., 2025)	Ensemble	2.635982
ThetaEvolve (Wang et al., 2025)	R1-Qwen3-8B	2.635983
TTT-Discover (Yuksekgonul et al., 2026)	Qwen3-8B	2.635983
Ours (REVES)	Qwen3-4B	2.635983

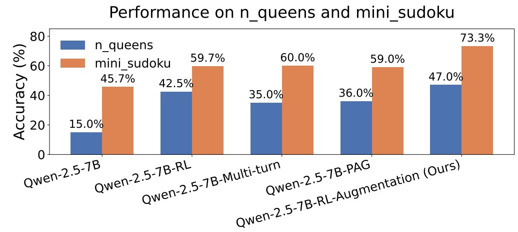


Figure 5: REVES performs well beyond standard reasoning benchmarks. **Left:** circle packing benchmark (sum of radii at $n = 26$, ↑); REVES on Qwen3-4B matches the best previously reported number despite using a smaller base model and less rollouts. **Right:** out-of-distribution puzzle benchmarks (n_queens and mini_sudoku) under verifier-based stopping.

Matching state-of-the-art on circle packing. We further evaluate REVES on the circle packing benchmark, where the goal is to pack n unit-area circles into a unit square to maximize the sum of their radii. We directly optimize Eq. (1) on this task. With a Qwen3-4B, our trained policy matches the previously reported best results from much larger evolutionary search systems built on Gemini-2.0 Pro/Flash and Qwen3-8B (Figure 5). Full implementation details are in Appendix H.

Table 2: Sequential-revision test-time scaling performance on math benchmarks (AIME24/25, MATH500). SelfConf-B and Oracle-B are sequential-revision (SR) test-time scaling with budget B, using the model's own confidence as a surrogate stopping rule and the ground-truth answer as the oracle stopping rule, respectively. † denotes evaluation using the authors' original codebase and protocol. Best and second-best results are shown in bold and underline.

Model	AIME 24				AIME 25				MATH 500			
	OneShot	Oracle-32	Oracle-4	SelfConf-4	OneShot	Oracle-32	Oracle-4	SelfConf-4	OneShot	Oracle-32	Oracle-4	SelfConf-4
Qwen-2.5-7B	2.03	13.96	4.79	2.34	0.26	6.25	1.25	0.57	23.83	50.05	28.98	26.10
Qwen-2.5-7B-RL	**17.86**	33.49	21.41	18.65	**10.35**	22.86	13.44	12.45	76.13	85.90	79.43	**78.60**
Qwen-2.5-7B-Multi-turn	14.58	30.26	20.00	13.80†	7.19	25.31	11.77	9.84†	75.93	87.13	80.13	77.95†
Qwen-2.5-7B-PAG	15.31	31.09	19.58	12.97†	7.45	26.82	12.81	9.43†	76.13	86.68	80.75	77.20†
Qwen-2.5-7B-RL-Aug	14.90	**45.73**	**25.94**	**19.58**	9.27	**40.52**	**22.34**	**15.10**	**76.75**	**94.68**	**85.60**	76.20
Qwen-2.5-3B	1.15	12.92	3.44	1.93	0.47	7.92	1.30	0.57	30.53	66.98	39.33	38.40
Qwen-2.5-3B-RL	6.41	16.93	8.13	7.40	1.82	9.01	3.91	2.40	64.93	80.53	69.57	65.63
Qwen-2.5-3B-Multi-turn	**7.60**	20.52	11.98	7.97†	2.60	12.71	5.16	2.29†	**65.28**	81.23	72.65	**67.00**†
Qwen-2.5-3B-PAG	6.30	14.79	7.92	5.31†	**3.44**	11.41	4.69	4.11†	64.00	78.75	69.90	65.63†
Qwen-2.5-3B-RL-Aug	6.20	**33.70**	**14.74**	**8.23**	2.76	**24.36**	**7.19**	**4.17**	64.47	**89.90**	**77.78**	66.03

Generalization to out-of-distribution puzzle tasks. As illustrated in Figure 5, we evaluate puzzle tasks using checkpoints trained only on math and code data, without any puzzle-specific training. Despite this distribution shift, RL-Augmentation substantially improves performance on out-of-distribution puzzle benchmarks, indicating strong cross-task generalization.

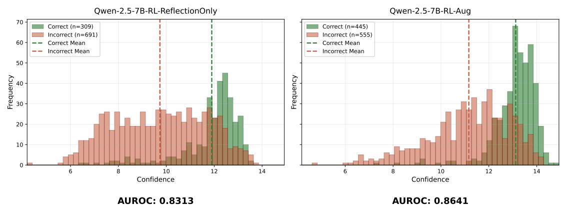
$J_{\phi_{\mathrm{SR}}}$-trained policies also improve other TTS algorithms. Table 3 shows that models trained with RL-Augmentation consistently improve the performance of a wide range of test-time algorithms that rely on sequential revision, including Best-of-N, MCTS (Inoue et al., 2025), AB-MCTS variants (Inoue et al., 2025) and Mind Evolution (Lee et al., 2025b), This suggests that sequential revision capability serves as a fundamental backbone that raises the performance ceiling of diverse test-time inference strategies.

Table 3: Comparison of model performance on different test-time strategies. Best results in each column are in bold; second-best are underlined.

Model	OneShot	SeqRev	BoN	MCTS	Mind Evolution	AB-MCTS-A (Beta)	AB-MCTS-A (Gaussian)	AB-MCTS-M
Qwen-2.5-7B-RL	18.02	23.04	27.82	22.67	25.74	25.98	27.33	26.23
Qwen-2.5-7B-Multi-turn	19.36	25.49	28.68	26.10	26.96	28.55	29.04	27.82
Qwen-2.5-7B-PAG	**19.49**	25.73	**30.39**	26.35	27.70	29.66	28.06	28.68
Qwen-2.5-7B-RL-Augmentation	18.87	**29.54**	29.29	**28.92**	**29.04**	**31.00**	**30.02**	**30.39**

Table 4: Effect of revision and verification prompts on AIME25. **Left:** OneShot, Oracle-4, and SelfConf-4 accuracy across the four configurations: *RL* (no augmentation), *VerificationOnly* (verification prompts only), *RevisionOnly* (revision prompts only), and *RL-Augmentation* (both, our method). RL-Augmentation dominates under Oracle-4 and SelfConf-4. **Right:** `TailConfidence` distributions for RevisionOnly and RL-Aug; RL-Augmentation produces sharper, better-separated correct/incorrect distributions, indicating improved confidence calibration.

Model	OneShot	Oracle-4	SelfConf-4
Qwen-2.5-7B-RL	**10.35**	13.44	12.45
Qwen-2.5-7B-VerificationOnly	6.25	10.26	7.08
Qwen-2.5-7B-RevisionOnly	7.86	16.46	8.33
Qwen-2.5-7B-RL-Augmentation	9.48	**21.16**	**15.83**



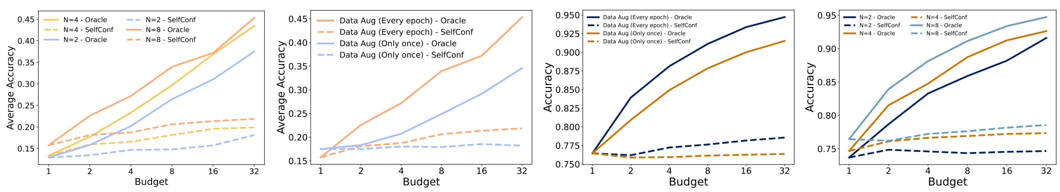


Figure 6: Ablations on data augmentation. From left to right: **(a)** effect of augmentation budget K (the maximum trajectory length allowed during augmentation) on AIME24; **(b)** continual versus one-shot augmentation on AIME24; **(c)** effect of augmentation frequency on MATH500; **(d)** effect of allowed revision length on MATH500. The MATH500 panels mirror the AIME24 trends, showing that the design choices generalize across datasets.

5.2 Ablation Study

The role of revision and verification prompts. As shown in Table 4, we investigate the roles of revision and verification prompts in sequential revision training. Qwen-2.5-7B-RevisionOnly augments RL-generated data using only revision prompts, while Qwen-2.5-7B-VerificationOnly augments data using only verification prompts. Qwen-2.5-7B-RL-Augmentation combines both prompts during data augmentation. Results under Oracle-4 indicate that revision prompts primarily improve *sequential revision capability*, whereas verification prompts do not directly enhance revision but instead train the model to assess solution validity.

To isolate the effect of verification prompts, we evaluate confidence calibration using Area Under the Receiver Operating Characteristic curve (AUROC) on AIME25. For each problem, we sample multiple model responses, label them as correct or incorrect based on the final answer, and compute the probability that a randomly sampled correct response receives a higher `TailConfidence` score than a randomly sampled incorrect one. The final AUROC is obtained by averaging over problems with sufficient correct samples. Under this evaluation, Qwen-2.5-7B-RL-Augmentation achieves a higher AUROC (74.1%) than RevisionOnly (72.1%), demonstrating that verification prompts improve confidence calibration. Thus, combining revision and verification prompts yields consistent gains by jointly enhancing revision capability and confidence estimation.

Necessity of continual data augmentation. As shown in Figure 6(b), augmenting data only at the initial epoch leads to substantially worse performance than continuous augmentation. This is because once the model stops making earlier mistakes, stale augmented data no longer provides informative learning signals, whereas regenerating data each epoch keeps the supervision aligned with the model's current failure modes.

Effect of augmentation budget. We study the effect of the augmentation budget on training performance. As shown in Figure 6(a), increasing the augmentation budget consistently improves performance, indicating that our method exhibits strong scaling behavior with respect to the augmentation budget. At the same time, allowing a larger augmentation length N naturally incurs higher computational cost, reflecting a trade-off between performance and compute in data augmentation.

References

Chen, Z., Qin, X., Wu, Y., Ling, Y., Ye, Q., Zhao, W. X. and Shi, G. (2025). Pass@k training for adaptively balancing exploration and exploitation of large reasoning models. https://arxiv.org/abs/2508.10751

Chow, Y., Tennenholtz, G., Gur, I., Zhuang, V., Dai, B., Thiagarajan, S., Boutilier, C., Agarwal, R., Kumar, A. and Faust, A. (2025). Inference-aware fine-tuning for best-of-n sampling in large language models. https://arxiv.org/abs/2412.15287

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z. and Zhang, Z. (2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. https://arxiv.org/abs/2501.12948

Fu, Y., Wang, X., Tian, Y. and Zhao, J. (2025). Deep think with confidence. https://arxiv.org/abs/2508.15260

Georgiev, B., Gómez-Serrano, J., Tao, T. and Wagner, A. Z. (2025). Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*.

Inoue, Y., Misaki, K., Imajuku, Y., Kuroki, S., Nakamura, T. and Akiba, T. (2025). Wider or deeper? scaling llm inference-time compute with adaptive branching tree search. https://arxiv.org/abs/2503.04412

Jain, A. K., Gonzalez-Pumariega, G., Chen, W., Rush, A. M., Zhao, W. and Choudhury, S. (2025). Multi-turn code generation through single-step rewards. https://arxiv.org/abs/2502.20380

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K. and Stoica, I. (2024). Livecodebench: Holistic and contamination free evaluation of large language models for code. https://arxiv.org/abs/2403.07974

Jiang, Y., Xiong, Y., Yuan, Y., Xin, C., Xu, W., Yue, Y., Zhao, Q. and Yan, L. (2025). Pag: Multi-turn reinforced llm self-correction with policy as generative verifier. https://arxiv.org/abs/2506.10406

Kumar, A., Zhuang, V., Agarwal, R., Su, Y., Co-Reyes, J. D., Singh, A., Baumli, K., Iqbal, S., Bishop, C., Roelofs, R., Zhang, L. M., McKinney, K., Shrivastava, D., Paduraru, C., Tucker, G., Precup, D., Behbahani, F. and Faust, A. (2024). Training language models to self-correct via reinforcement learning. https://arxiv.org/abs/2409.12917

Lange, R. T., Imajuku, Y. and Cetin, E. (2025). Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*.

Lee, H., Oh, S., Kim, J., Shin, J. and Tack, J. (2025a). Revise: Learning to refine at test-time via intrinsic self-verification.
https://arxiv.org/abs/2502.14565

Lee, K.-H., Fischer, I., Wu, Y.-H., Marwood, D., Baluja, S., Schuurmans, D. and Chen, X. (2025b). Evolving deeper llm thinking.
https://arxiv.org/abs/2501.09891

Li, A., Wang, Y., Yuan, Z., Jegelka, S. and Wang, Y. (2025a). Lanpo: Bootstrapping language and numerical feedback for reinforcement learning in llms.
https://arxiv.org/abs/2510.16552

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Lago, A. D., Hubert, T., Choy, P., de Masson d'Autume, C., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., Molloy, J., Mankowitz, D. J., Robson, E. S., Kohli, P., de Freitas, N., Kavukcuoglu, K. and Vinyals, O. (2022). Competition-level code generation with alphacode. *Science*, **378** 1092–1097.
https://www.science.org/doi/abs/10.1126/science.abq1158

Li, Y., Lyu, M. and Wang, L. (2025b). Learning to reason from feedback at test-time.
https://arxiv.org/abs/2502.15771

Liang, X., Li, Z.-Z., Gong, Y., Wang, Y., Zhang, H., Shen, Y., Wu, Y. N. and Chen, W. (2025). Sws: Self-aware weakness-driven problem synthesis in reinforcement learning for llm reasoning.
https://arxiv.org/abs/2506.08989

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A. and Clark, P. (2023). Self-refine: Iterative refinement with self-feedback.
https://arxiv.org/abs/2303.17651

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J., Mehrabian, A. et al. (2025). Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J. and Lowe, R. (2022). Training language models to follow instructions with human feedback.
https://arxiv.org/abs/2203.02155

Qu, Y., Setlur, A., Smith, V., Salakhutdinov, R. and Kumar, A. (2026). Pope: Learning to reason on hard problems via privileged on-policy exploration.
https://arxiv.org/abs/2601.18779

Qu, Y., Zhang, T., Garg, N. and Kumar, A. (2024). Recursive introspection: Teaching language model agents how to self-improve.
https://arxiv.org/abs/2407.18219

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K. and Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning.
https://arxiv.org/abs/2303.11366

Stojanovski, Z., Stanley, O., Sharratt, J., Jones, R., Adefioye, A., Kaddour, J. and Köpf, A. (2025). Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards.
https://arxiv.org/abs/2505.24760

Tang, Y., Zheng, K., Synnaeve, G. and Munos, R. (2025). Optimizing language models for inference time objectives using reinforcement learning.
https://arxiv.org/abs/2503.19595

Walder, C. and Karkhanis, D. (2025). Pass@k policy optimization: Solving harder reinforcement learning problems.
https://arxiv.org/abs/2505.15201

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A. and Zhou, D. (2023). Self-consistency improves chain of thought reasoning in language models.
https://arxiv.org/abs/2203.11171

Wang, Y., Su, S.-R., Zeng, Z., Xu, E., Ren, L., Yang, X., Huang, Z., He, X., Ma, L., Peng, B. et al. (2025). Thetaevolve: Test-time learning on open problems. *arXiv preprint arXiv:2511.23473*.

Xie, J., Zhang, K., Chen, J., Zhu, T., Lou, R., Tian, Y., Xiao, Y. and Su, Y. (2024). Travelplanner: A benchmark for real-world planning with language agents.
https://arxiv.org/abs/2402.01622

Xiong, W., Shi, C., Shen, J., Rosenberg, A., Qin, Z., Calandriello, D., Khalman, M., Joshi, R., Piot, B., Saleh, M., Jin, C., Zhang, T. and Liu, T. (2025). Building math agents with multi-turn iterative preference learning.
https://arxiv.org/abs/2409.02392

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z. and Qiu, Z. (2025). Qwen3 technical report.
https://arxiv.org/abs/2505.09388

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., Dong, G., Wei, H., Lin, H., Tang, J., Wang, J., Yang, J., Tu, J., Zhang, J., Ma, J., Yang, J., Xu, J., Zhou, J., Bai, J., He, J., Lin, J., Dang, K., Lu, K., Chen, K., Yang, K., Li, M., Xue, M., Ni, N., Zhang, P., Wang, P., Peng, R., Men, R., Gao, R., Lin, R., Wang, S., Bai, S., Tan, S., Zhu, T., Li, T., Liu, T., Ge, W., Deng, X., Zhou, X., Ren, X., Zhang, X., Wei, X., Ren, X., Liu, X., Fan, Y., Yao, Y., Zhang, Y., Wan, Y., Chu, Y., Liu, Y., Cui, Z., Zhang, Z., Guo, Z. and Fan, Z. (2024). Qwen2 technical report.
https://arxiv.org/abs/2407.10671

Yuksekgonul, M., Koceja, D., Li, X., Bianchi, F., McCaleb, J., Wang, X., Kautz, J., Choi, Y., Zou, J., Guestrin, C. and Sun, Y. (2026). Learning to discover at test time.
https://arxiv.org/abs/2601.16175

Zhang, X., Zhang, Y., Sun, H., Feng, K., Lu, C., Yang, C. and Meng, H. (2026). Critique-grpo: Advancing llm reasoning with natural language and numerical feedback.
https://arxiv.org/abs/2506.03106

A Sequential Revision Protocol

In this section, we elaborate on the test-time protocol described in Protocol 1 and illustrate its practical usage with several concrete examples, including a mathematical reasoning task in Appendix A.1 and a coding task in Appendix A.2.

Protocol 1 Sequential Revision Protocol

Input: revision budget K, question x, model π
Sample initial response $y_1 \sim \pi(\cdot \mid x)$
for $i = 2$ **to** K **do**
  Generate revised response $y_i \sim \pi(\cdot \mid x, y_{i-1}, \texttt{Feedback}(y_{i-1}))$
  if $r^\star(x, y_i) = 1$ **then**
    # Early stopping
    Set $\hat{y} = y_i$
    break for loop
  end if
end for
if No early stopping **then**
  $\hat{y} = y_K$
end if

A.1 Example Math Task

In mathematical tasks, the feedback consists solely of the sentence "The response is incorrect." (marked in red in the figure), without offering further diagnostic information.

Input (Turn 1)

Solve the following math problem. Show brief reasoning and provide the final answer.
Problem: A 90° rotation around $-4 - 5i$ in the clockwise direction is applied to 3 (as a complex number). What is the resulting complex number?

LLM Output (Turn 1)

To solve this problem, we need to perform a 90° clockwise rotation around the complex number $-4 - 5i$.
First, it's useful to understand that a 90° clockwise rotation of a complex number z around another complex number a is equivalent to the following transformation:

$$z' = a + (a - z) \cdot i$$

However, since we are rotating around $-4 - 5i$, let's denote this center of rotation as $a = -4 - 5i$. Our complex number $z = 3$ needs to be rotated.
1. Find the difference between z and a:

$$z - a = 3 - (-4 - 5i) = 3 + 4 + 5i = 7 + 5i$$

2. Perform the 90° clockwise rotation by multiplying this difference by i and adding it back to a:

$$z' = a + (z - a) \cdot i = (-4 - 5i) + (7 + 5i) \cdot i$$

Next, simplify the expression:

$$(7 + 5i) \cdot i = 7i + 5i^2$$

Since $i^2 = -1$:

$$7i + 5(-1) = 7i - 5$$

$$\ldots$$

Therefore, the final answer is $\boxed{-9 + 2i}$.

Input (Turn 2)

Problem: A 90° rotation around $-4-5i$ in the clockwise direction is applied to 3 (as a complex number). What is the resulting complex number?
Previous Attempt: To solve this problem, we need to perform a 90° clockwise rotation around the complex number $-4-5i$.
First, it's useful to understand that a 90° clockwise rotation of a complex number z around another complex number a is equivalent to the following transformation:

$$z' = a + (a - z) \cdot i$$

However, since we are rotating around $-4-5i$, let's denote this center of rotation as $a = -4-5i$. Our complex number $z = 3$ needs to be rotated.
1. Find the difference between z and a:

$$z - a = 3 - (-4-5i) = 3 + 4 + 5i = 7 + 5i$$

2. Perform the 90° clockwise rotation by multiplying this difference by i and adding it back to a:

$$z' = a + (z - a) \cdot i = (-4-5i) + (7+5i) \cdot i$$

Next, simplify the expression:

$$(7+5i) \cdot i = 7i + 5i^2$$

Since $i^2 = -1$:

$$7i + 5(-1) = 7i - 5$$

$$\dots$$

So, the resulting complex number after the rotation is $-9+2i$.
Therefore, the final answer is $\boxed{-9+2i}$.
Verifiable Feedback: The response is incorrect.
Now produce a corrected solution. Keep reasoning concise.

LLM Output (Turn 2)

To correctly rotate a complex number 3 around $-4-5i$ by 90° clockwise, we need to use the appropriate transformation for rotations in the complex plane.
Given a complex number $z = 3$, and the center of rotation $a = -4-5i$, a 90° clockwise rotation of z around a can be represented as:

$$z' = a + (a - z) \cdot (-i)$$

Let's go through the steps in detail:
1. **Calculate $a - z$:**

$$a - z = (-4-5i) - 3 = -4 - 5i - 3 = -7 - 5i$$

2. **Multiply by $-i$:**

$$(a - z) \cdot (-i) = (-7-5i) \cdot (-i) = (-7)(-i) + (-5i)(-i) = 7i + 5i^2$$

Since $i^2 = -1$:

$$7i + 5(-1) = 7i - 5$$

3. **Add this result back to a:**

$$z' = a + (7i - 5) = (-4-5i) + (-5+7i)$$

$$z' = -4 - 5i - 5 + 7i$$

Combine the real and imaginary parts:

$$z' = (-4-5) + (-5i + 7i) = -9 + 2i$$

Thus, after performing the 90° clockwise rotation, the resulting complex number is $\boxed{-9+2i}$.

Input (Turn 3)

Problem: A $90°$ rotation around $-4-5i$ in the clockwise direction is applied to 3 (as a complex number). What is the resulting complex number?
Previous Attempt: To correctly rotate a complex number 3 around $-4-5i$ by $90°$ clockwise, we need to use the appropriate transformation for rotations in the complex plane.
Given a complex number $z = 3$, and the center of rotation $a = -4 - 5i$, a $90°$ clockwise rotation of z around a can be represented as:

$$z' = a + (a - z) \cdot (-i)$$

Let's go through the steps in detail:
1. **Calculate $a - z$:**

$$a - z = (-4 - 5i) - 3 = -4 - 5i - 3 = -7 - 5i$$

2. **Multiply by $-i$:**

$$(a - z) \cdot (-i) = (-7 - 5i) \cdot (-i) = (-7)(-i) + (-5i)(-i) = 7i + 5i^2$$

Since $i^2 = -1$:

$$7i + 5(-1) = 7i - 5$$

3. **Add this result back to a:**

$$z' = a + (7i - 5) = (-4 - 5i) + (-5 + 7i)$$

$$z' = -4 - 5i - 5 + 7i$$

Combine the real and imaginary parts:

$$z' = (-4 - 5) + (-5i + 7i) = -9 + 2i$$

Thus, after performing the $90°$ clockwise rotation, the resulting complex number is $\boxed{-9 + 2i}$.
Verifiable Feedback: The response is incorrect.
Now produce a corrected solution. Keep reasoning concise.

LLM Output (Turn 3)

To find the result after a $90°$ clockwise rotation around $-4-5i$, we can use the following transformation steps. We start by considering a complex number $z = 3$ and the center of rotation $a = -4 - 5i$:
1. **Move z to the origin relative to a:**

$$z - a = 3 - (-4 - 5i) = 3 + 4 + 5i = 7 + 5i$$

2. **Rotate by $90°$ clockwise:** A $90°$ clockwise rotation of a complex number w is equivalent to multiplying w by $-i$. So, we need to multiply $7 + 5i$ by $-i$:

$$(7 + 5i) \times (-i) = 7 \times (-i) + 5i \times (-i) = -7i - 5i^2$$

Since $i^2 = -1$:

$$-7i - 5(-1) = -7i + 5 = 5 - 7i$$

3. **Translate back by adding a:** Adding a to the resultant, we get:

$$z' = (5 - 7i) + (-4 - 5i) = 5 - 4 - 7i - 5i = 1 - 12i$$

Thus, the resulting complex number after the $90°$ clockwise rotation is $\boxed{1 - 12i}$.

A.2 Example Coding Task

At test time for coding tasks, we follow an iterative execution-based evaluation protocol. The model first generates a candidate program, which is then executed against the publicly available test cases provided by the problem. If the program fails any test, the resulting error messages or failing test feedback are collected and appended to the context for the next generation step, allowing the model to revise its solution based on execution feedback. Note that passing the public test cases does not guarantee correctness: a solution is considered correct only if it passes all test cases, including both public and private ones.

Input (Turn 1)

You are an expert Python programmer. You will be given a problem specification and must produce a correct Python program that passes the tests.

Question: You are given a permutation $P = (P_1, P_2, \dots, P_N)$ of $(1, 2, \dots, N)$. You want to satisfy $P_i = i$ for all $i = 1, 2, \dots, N$ by performing the following operation zero or more times:

- Choose an integer k such that $1 \leq k \leq N$. If $k \geq 2$, sort the 1-st through $(k-1)$-th terms of P in ascending order. Then, if $k \leq N - 1$, sort the $(k+1)$-th through N-th terms of P in ascending order.

It can be proved that under the constraints of this problem, it is possible to satisfy $P_i = i$ for all $i = 1, 2, \dots, N$ with a finite number of operations for any P. Find the minimum number of operations required. You have T test cases to solve.

Input ...

Output
Print T lines. The i-th line should contain the answer for the i-th test case.

Constraints ...

Sample Input 1

- `3`
- `5`
- ...

Sample Output 1

- `1`
- ...

Format: Read inputs from stdin and write the answer to stdout. Enclose your final code in the following box.

```
# YOUR CODE HERE
```

Answer: Provide *only* the final Python code.

LLM Output (Turn 1)

To solve this problem, we aim to determine the minimum number of operations required to transform a given permutation P into the identity permutation $(1, 2, \dots, N)$. The key insight is that each operation can be viewed as a global reordering step that partially resolves disorder in the permutation. Intuitively, such operations progressively eliminate inconsistencies between positions and values.
...

```
def min_operations(N, P):
    count = 0
    for i in range(N):
        if P[i] != i + 1:
            count += 1
    return count

...

for _ in range(T):
    N = int(input())
    P = list(map(int, input().split()))
    print(min_operations(N, P))
```

Input (Turn 2)

You are an expert Python programmer. Given a problem specification, produce a correct Python program that passes all tests.

Question: Given a permutation $P = (P_1, \ldots, P_N)$ of $(1, \ldots, N)$, the goal is to transform P into the identity permutation . . .

Input / Output. . . .

Previous Codes

code_v1

```
# Your task is to find the minimum number of operations required
to sort the permutation P.

def min_operations(N, P):
    operations = 0
    for i in range(N):
        while P[i] != i + 1:
            ...
    N = case[0]
    P = case[1:]
    print(min_operations(N, P))
```

Issues with code_v1

For the input

$$\begin{array}{l} 3 \\ 5 \\ 2\ 1\ 3\ 5\ 4 \\ 3 \\ 1\ 2\ 3 \\ 7 \\ 3\ 2\ 1\ 7\ 5\ 6\ 4 \end{array}$$

the expected output is

$$\begin{array}{l} 1 \\ 0 \\ 2 \end{array}$$

but `code_v1` results in a runtime error. The failure is primarily due to incorrect handling of index-based updates and repeated linear searches (`P.index`), which lead to inconsistent state updates and excessive runtime overhead.

LLM Output (Turn 2)

To solve this problem, we aim to determine the minimum number of operations required to transform a given permutation P into the identity permutation $(1, 2, \ldots, N)$. The key insight is that each operation can be viewed as a global reordering step that partially resolves disorder in the permutation. Intuitively, such operations progressively eliminate inconsistencies between positions and values. . . .

```
def min_operations(N, P):
    count = 0
    for i in range(N):
        if P[i] != i + 1:
            count += 1
    return count

...

for _ in range(T):
    N = int(input())
    P = list(map(int, input().split()))
    print(min_operations(N, P))
```

B Limitation and Future Direction

Limitation. REVES requires a ground-truth verifier during training. Math, coding, puzzle, and circle-packing benchmarks all admit deterministic verifiers, so REVES applies directly. For tasks without such verifiers (free-form QA, creative writing, RLHF-style preference learning), REVES does not directly extend; the verification prompts trained alongside revision data can be repurposed for self-verification at test time, but their reliability depends on the policy's own calibration.

Future direction. The verifier requirement points to a natural class of problems where REVES is especially well suited: scientific tasks whose objectives admit a clear, automated reward. Examples include solving partial differential equations (PDEs), where residual norms or analytic checks serve as verifiers; symbolic regression, where fit and complexity supply a numeric score; theorem proving, where a proof checker provides binary feedback; and discovery-style optimization tasks like circle packing (Appendix H), where the verifier is the geometric solver itself. In all of these settings, the cost of producing the verifier signal is offset by an essentially unlimited supply of training problems and a precise notion of correctness, both of which line up well with REVES's reliance on per-state one-step recovery as the gradient signal. Scaling REVES to such scientific-discovery domains is a natural next step.

C Proofs

C.1 Proof of Theorem 3.2

Theorem C.1 (Objective mismatch). *For any revision length $K \geq 2$, there exist policies π_1, π_2 and a problem distribution $\mathcal{X}$ such that*

$$J_{\text{OneShot}}(\pi_1) = J_{\text{OneShot}}(\pi_2), \qquad J_{\phi_{\text{SR}}}(\pi_2) - J_{\phi_{\text{SR}}}(\pi_1) \geq \Delta(K),$$

where $\Delta(K) > 0$.

Proof. **Construction.** Take $\mathcal{X}$ supported on a single problem x with $N \geq 2$ candidate answers $\{y^{(1)}, \dots, y^{(N)}\}$ and ground-truth $y^\star$ drawn uniformly from this set; feedback is binary correctness. Define two history-independent policies, each sampling from a fixed distribution at every step:

$$\pi_1(\cdot \mid \cdot) \;=\; \delta_{y^{(1)}}, \qquad \pi_2(\cdot \mid \cdot) \;=\; \text{Uniform}\{y^{(1)}, \dots, y^{(N)}\},$$

with π_2's draws independent across steps.

One-shot. Since $y^\star$ is uniform,

$$J_{\text{OneShot}}(\pi_1) \;=\; \mathbb{P}(y^\star = y^{(1)}) \;=\; \tfrac{1}{N}, \qquad J_{\text{OneShot}}(\pi_2) \;=\; \sum_{i=1}^{N} \tfrac{1}{N} \cdot \tfrac{1}{N} \;=\; \tfrac{1}{N},$$

so $J_{\text{OneShot}}(\pi_1) = J_{\text{OneShot}}(\pi_2)$.

Sequential revision (budget K). π_1 outputs $y^{(1)}$ at every step and therefore succeeds iff $y^\star = y^{(1)}$, giving $J_{\phi_{\text{SR}}}(\pi_1) = 1/N$. For π_2, the K i.i.d. uniform draws are independent of $y^\star$, so the probability that none equals $y^\star$ is $(1 - 1/N)^K$, giving $J_{\phi_{\text{SR}}}(\pi_2) = 1 - (1 - 1/N)^K$.

Gap. For $K \geq 2$, monotonicity of $K \mapsto (1 - 1/N)^K$ on $K \geq 1$ gives $(1 - 1/N)^K \leq (1 - 1/N)^2 = 1 - 2/N + 1/N^2$, hence

$$\Delta(K) \;:=\; J_{\phi_{\text{SR}}}(\pi_2) - J_{\phi_{\text{SR}}}(\pi_1) \;=\; 1 - \left(1 - \tfrac{1}{N}\right)^K - \tfrac{1}{N} \;\geq\; \tfrac{N-1}{N^2} \;>\; 0. \qquad \square$$

C.2 Proof of Theorem 3.1

The argument uses the recovery decomposition $J_\phi(\pi) = \sum_z \rho_\pi^\phi(z)\, V_\pi(z)$, which holds for any first-success transition strategy ϕ paired with any policy π. The proof of Lemma 4.1 (Appendix C.3) establishes this for $\phi = \phi_{\text{SR}}$; the same three steps – (i) writing the success event at step t as $\{\tau \geq t,\, y_t \in G_x\}$, (ii) Fubini against $\sum_{t=1}^{K} \mathbf{1}\{\tau \geq t\} = \tau$, (iii) commuting the visit-count sum with the outer expectation – depend only on the first-success stopping rule and the Markov property of π, so they generalize verbatim from $\rho_\pi^{\phi_{\text{SR}}}$ to ρ_π^ϕ for any $\phi \in \Phi_{\mathcal{R}}$ that early-stops.

Proof of Theorem 3.1. By the recovery decomposition applied to π_1 and π_0,

$$J_\phi(\pi_1) - J_\phi(\pi_0) = \sum_z \rho^\phi_{\pi_1}(z)\, V_{\pi_1}(z) - \sum_z \rho^\phi_{\pi_0}(z)\, V_{\pi_0}(z).$$

Add and subtract $\sum_z \rho^\phi_{\pi_0}(z)\, V_{\pi_1}(z)$ and group:

$$J_\phi(\pi_1) - J_\phi(\pi_0) = \underbrace{\sum_z \rho^\phi_{\pi_0}(z)\big(V_{\pi_1}(z) - V_{\pi_0}(z)\big)}_{(\star)} + \underbrace{\sum_z \big(\rho^\phi_{\pi_1}(z) - \rho^\phi_{\pi_0}(z)\big)\, V_{\pi_1}(z)}_{(\dagger)}. \tag{2}$$

Bounding $(\dagger)$**.** Since $V_{\pi_1}(z) \in [0,1]$,

$$(\dagger) \;\geq\; -\sum_z \big|\rho^\phi_{\pi_1}(z) - \rho^\phi_{\pi_0}(z)\big| \;=\; -\big\|\rho^\phi_{\pi_1} - \rho^\phi_{\pi_0}\big\|_1. \tag{3}$$

Bounding $(\star)$**.** We claim the pointwise inequality

$$\rho^\phi_{\pi_0}(z)\big(V_{\pi_1}(z) - V_{\pi_0}(z)\big) \;\geq\; \frac{1}{C_\phi}\, \rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z)\big(V_{\pi_1}(z) - V_{\pi_0}(z)\big) \quad \text{for every } z. \tag{4}$$

Case 1: $z \in \operatorname{supp}(\rho^\phi_{\pi_0})$. By (C2), $V_{\pi_1}(z) - V_{\pi_0}(z) \geq 0$. By (C1), $\rho^\phi_{\pi_0}(z) \geq C_\phi^{-1}\, \rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z) \geq 0$. Multiplying both sides of (C1) by the nonnegative quantity $V_{\pi_1}(z) - V_{\pi_0}(z)$ preserves the inequality and gives (4).

Case 2: $z \notin \operatorname{supp}(\rho^\phi_{\pi_0})$. Then $\rho^\phi_{\pi_0}(z) = 0$, and (C1) forces $0 \geq C_\phi^{-1}\, \rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z)$. Combined with $\rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z) \geq 0$, this gives $\rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z) = 0$, so both sides of (4) vanish.

Summing (4) over z,

$$(\star) \;\geq\; \frac{1}{C_\phi} \sum_z \rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z)\big(V_{\pi_1}(z) - V_{\pi_0}(z)\big). \tag{5}$$

Converting (5) **to** $J_{\phi_{\mathrm{SR}}}(\pi_1) - J_{\phi_{\mathrm{SR}}}(\pi_0)$**.** Apply the recovery decomposition to $\phi = \phi_{\mathrm{SR}}$ at π_1 and π_0, then add and subtract $\sum_z \rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z)\, V_{\pi_1}(z)$:

$$J_{\phi_{\mathrm{SR}}}(\pi_1) - J_{\phi_{\mathrm{SR}}}(\pi_0) = \sum_z \rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z)\big(V_{\pi_1}(z) - V_{\pi_0}(z)\big) + \sum_z \big(\rho^{\phi_{\mathrm{SR}}}_{\pi_1}(z) - \rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z)\big)\, V_{\pi_1}(z).$$

Rearranging and using $V_{\pi_1}(z) \in [0,1]$,

$$\sum_z \rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z)\big(V_{\pi_1}(z) - V_{\pi_0}(z)\big) \;\geq\; \big[J_{\phi_{\mathrm{SR}}}(\pi_1) - J_{\phi_{\mathrm{SR}}}(\pi_0)\big] - \big\|\rho^{\phi_{\mathrm{SR}}}_{\pi_1} - \rho^{\phi_{\mathrm{SR}}}_{\pi_0}\big\|_1. \tag{6}$$

Combine. Chaining (6) into (5) and adding (3),

$$J_\phi(\pi_1) - J_\phi(\pi_0) \;\geq\; \frac{1}{C_\phi}\big[J_{\phi_{\mathrm{SR}}}(\pi_1) - J_{\phi_{\mathrm{SR}}}(\pi_0)\big] \;-\; \big\|\rho^\phi_{\pi_1} - \rho^\phi_{\pi_0}\big\|_1 \;-\; \frac{1}{C_\phi}\big\|\rho^{\phi_{\mathrm{SR}}}_{\pi_1} - \rho^{\phi_{\mathrm{SR}}}_{\pi_0}\big\|_1. \quad \square$$

Connection to our method. The third term $C_\phi^{-1}\, \|\rho^{\phi_{\mathrm{SR}}}_{\pi_1} - \rho^{\phi_{\mathrm{SR}}}_{\pi_0}\|_1$ in the bound is the price for placing the on-policy SR test-time objective $J_{\phi_{\mathrm{SR}}}(\pi) = \sum_z \rho^{\phi_{\mathrm{SR}}}_\pi(z)\, V_\pi(z)$ on the right-hand side: as the policy improves, SR's own visit measure $\rho^{\phi_{\mathrm{SR}}}_\pi$ also shifts, and that shift contaminates $J_{\phi_{\mathrm{SR}}}(\pi_1) - J_{\phi_{\mathrm{SR}}}(\pi_0)$. The off-policy fixed-reference quantity

$$J^{\mathcal{R}}_{\phi_{\mathrm{SR}}}(\pi) \;:=\; \sum_z \rho^{\phi_{\mathrm{SR}}}_{\pi_0}(z)\, V_\pi(z)$$

absorbs this shift by holding the visit measure at the baseline; $J^{\mathcal{R}}_{\phi_{\mathrm{SR}}}(\pi_1) - J^{\mathcal{R}}_{\phi_{\mathrm{SR}}}(\pi_0) = \sum_z \rho^{\phi_{\mathrm{SR}}}_{\pi_0}(\cdot)(V_{\pi_1} - V_{\pi_0})$ exactly, and (5) terminates the proof in one step:

$$J_\phi(\pi_1) - J_\phi(\pi_0) \;\geq\; \frac{1}{C_\phi}\big[J^{\mathcal{R}}_{\phi_{\mathrm{SR}}}(\pi_1) - J^{\mathcal{R}}_{\phi_{\mathrm{SR}}}(\pi_0)\big] - \big\|\rho^\phi_{\pi_1} - \rho^\phi_{\pi_0}\big\|_1.$$

This off-policy form is exactly the training signal REVES optimizes (Section 4): Stage I freezes the rollout distribution at the current π_0, generating SR trajectories whose visited states constitute samples from $\rho^{\phi_{\mathrm{SR}}}_{\pi_0}$; Stage II then trains V_π on those fixed states via single-turn RL. Each epoch's rollout refresh resets the baseline so the ℓ_1 shift on $\rho^{\phi_{\mathrm{SR}}}$ stays small. The two forms are mathematically equivalent; the off-policy form makes the algorithmic interpretation transparent.

Interpretation. The bound is conditional, not unconditional. It bites when (i) SR's revision-input distribution covers ϕ's (small C_ϕ in (C1)), (ii) the policy update raises one-step recovery on those states ((C2)), and (iii) the visit measures ρ^ϕ and $\rho^{\phi_{\rm SR}}$ are stable under the update (small ℓ_1 shifts). The empirical confirmation that $J_{\phi_{\rm SR}}$-trained policies improve other revision-using TTS algorithms (tree search, BoN-with-revision, evolutionary refinement) is in Table 3.

C.3 Proof of Lemma 4.1

Proof. We prove the three identities in (1) in order.

First identity. SR stops at the first success, so $r^\star(x, y_\tau) = 1 \iff \exists\, t \le K : r^\star(x, y_t) = 1$, and the success event at step t equals $\{\tau \ge t,\, y_t \in G_x\}$, where $G_x := \{y : r^\star(x, y) = 1\}$. By the tower property and the fact that π_θ is Markov in z_t,

$$\mathbb{P}(\tau \ge t,\, y_t \in G_x) = \mathbb{E}\big[\mathbf{1}\{\tau \ge t\}\, \mathbb{P}(y_t \in G_x \mid z_t)\big] = \mathbb{E}\big[\mathbf{1}\{\tau \ge t\}\, V_\pi(z_t)\big].$$

Summing over $t \le K$,

$$J_{\phi_{\rm SR}}(\theta) = \mathbb{P}\big(\exists\, t \le K : r^\star(x, y_t) = 1\big) = \sum_{t=1}^{K} \mathbb{E}\big[\mathbf{1}\{\tau \ge t\}\, V_\pi(z_t)\big].$$

Second identity. Using $\sum_{t=1}^{K} \mathbf{1}\{\tau \ge t\} = \tau$ and Fubini,

$$\sum_{t=1}^{K} \mathbb{E}\big[\mathbf{1}\{\tau \ge t\}\, V_\pi(z_t)\big] = \mathbb{E}\Big[\sum_{t=1}^{\tau} V_\pi(z_t)\Big].$$

Third identity. Substituting the definition $V_\pi(z) = \mathbb{E}_{y' \sim \pi_\theta(\cdot|z)}[r^\star(x, y')]$ and exchanging the visit-count sum with the outer expectation gives

$$\mathbb{E}\Big[\sum_{t=1}^{\tau} V_\pi(z_t)\Big] = \sum_z V_\pi(z)\, \mathbb{E}\Big[\sum_{t=1}^{\tau} \mathbf{1}\{z_t = z\}\Big] = \sum_z \rho_\theta(z)\, V_\pi(z) = \sum_z \rho_\theta(z)\, \mathbb{E}_{y' \sim \pi_\theta(\cdot|z)}\big[r^\star(x, y')\big].$$

□

D Training Prompt Template

In this section, we present the training prompt templates used for data augmentation. An important distinction from the test-time setting is that we do not append execution error information to the context during training. Instead, as illustrated in Figure 10 and Figure 11, the training prompts rely on structured supervision rather than explicit error feedback.

E Implementation Details

In this section, we provide details of the training and evaluation implementations used in our experiments. All experiments are conducted on machines equipped with 8 × NVIDIA H200 GPUs.

RL Baseline. Our RL baseline is implemented using the official `Skywork-OR1` codebase. We directly adopt their provided training scripts and configurations, which optimize a single-shot response using reinforcement learning. No revision or multi-turn generation is involved during training for this baseline.

Multi-Turn and PAG Baselines. The Multi-Turn and PAG baselines are implemented based on the `Policy-As-GenVerifier` codebase. To ensure comparability with our setting, we make two modifications to the original implementation. First, we increase the maximum response length to 8092 tokens to match our training setup. Second, we adapt the reward computation function so that it correctly evaluates responses under our datasets, while keeping the reward definition itself identical to that used in our methods. Apart from these changes, the original training pipelines and optimization settings are preserved.

Prompt Template for Revision in Math Tasks

Problem

A frequency distribution of the scores for Mr. Sampson's algebra class is shown. What percent of the class received a score in the $60\% - 69\%$ range?

Test Scores	Frequencies
$90\% - 100\%$	IIII
$80\% - 89\%$	IIII IIII
$70\% - 79\%$	IIII II
$60\% - 69\%$	IIII I
Below 60%	II

Previous Answer

To solve this problem, we first count the number of students in each score range. The total number of students is computed as $4 + 8 + 6 + 5 + 2 = 25$. The number of students scoring in the $60\% - 69\%$ range is 5. ...

Verifiable Feedback:

The previous answer is incorrect.

Instruction:

Now produce a corrected solution. Identify and fix potential mistakes in the previous attempt. Keep the reasoning concise and provide a corrected final answer. Do not assume access to the correct answer.

Figure 7: Prompt template for the revision stage with binary correctness feedback. Concrete problem content and partial incorrect answers are shown for illustration, with remaining content omitted for brevity.

Prompt Template for Verification in Math Tasks

Problem

A frequency distribution of the scores for Mr. Sampson's algebra class is shown. What percent of the class received a score in the $60\% - 69\%$ range?

Test Scores	Frequencies
$90\% - 100\%$	IIII
$80\% - 89\%$	IIII IIII
$70\% - 79\%$	IIII II
$60\% - 69\%$	IIII I
Below 60%	II

Proposed Answer

To solve this problem, we first identify the number of students in each score range from the frequency distribution. The total number of students is computed as $4 + 8 + 6 + 5 + 2 = 25$. Among them, 5 students fall into ...

Instruction:

Judge whether the proposed answer is correct. Respond concisely and end the final line with `VERDICT:` `True` or `VERDICT:` `False`.

Figure 8: Prompt template used for the verification stage. The verifier observes the problem and a proposed solution and outputs a binary correctness judgment.

RL-Augmentation (Ours). Our method builds on top of the same training framework as the baselines. The key difference lies in the data augmentation procedure. Before training, we first perform an offline augmentation pass to initialize the augmented dataset. During training, at the beginning of each epoch, we further augment the dataset by generating new revision trajectories using the current policy, and the newly generated samples are added to the training data. When generating augmented data, we allow a maximum revision budget of $N = 8$. The training prompts used for augmentation follow the templates described in Appendix D. The model is then trained on the mixture of original and augmented data using the same RL objective as the baseline.

Evaluation Protocol. Unless otherwise specified, all experiments are evaluated using our unified evaluation framework, which will be publicly released upon paper acceptance. Given an input problem, the model follows the corresponding test-time protocol (e.g., sequential revision when applicable) to generate a final response. The generated output is serialized into a JSON file and evaluated by an external, task-specific verifier. This pipeline is applied uniformly across all tasks and methods to ensure fair comparison: generation and evaluation are fully decoupled, and correctness is determined solely by the external verifier rather than any internal model signals. During evaluation, the maximum response length is set to 16,384 tokens for coding tasks and AIME24/25, and 8,096 tokens for MATH500. The decoding temperature is fixed to 0.6 for coding tasks and 1.0 for all other tasks, and the confidence threshold in the SelfConf setting is set to $c = 0.5$.

For math tasks, when evaluating the Multi-Turn and PAG baselines, we follow the original evaluation implementation provided in the `Policy-As-GenVerifier` repository to remain faithful to their protocol. Under this evaluation pipeline, we use the same maximum response length settings for 7B models as described above. For 3B models with revision budget $K = 4$, where the effective sequence length exceeds the model context limit, we reduce the maximum response length to 4,096 tokens. All math evaluations are conducted using `math_eval`, ensuring that results are directly comparable across evaluation settings.

F Method Details: Multi-Turn Comparison, Efficiency, and Test-Time Sensitivity

F.1 Comparison with Naive Multi-Turn RL

A natural alternative to our offline scheme is online sequential revision with standard multi-turn credit assignment. Two such schemes appear in the recent literature. The first, *trajectory-level broadcast*, samples G trajectories of M turns per prompt, scores each trajectory by its final reward, computes a trajectory-level advantage, and broadcasts that single advantage uniformly to every turn in the trajectory. The second, *pooled per-response*, pools all $G \times M$ responses across the G trajectories, treats each response as an independent sample, and computes advantages within the pool.

Neither scheme suits our setting. Sequential revision is structurally different from a vanilla multi-turn task: each turn is a fresh attempt at the *same* problem x, not a step in a longer-horizon decision process with intermediate sub-rewards. Trajectory-level broadcast assigns identical credit to every turn and therefore ignores that some intermediate responses are much closer to correctness than others. Pooled per-response also fails to capture this structure: the wrong intermediate responses are treated identically, even though the near-miss failures are exactly the ones we most want the policy to learn from. Our offline selection-then-replay scheme bypasses both pathologies. We identify near-miss failures explicitly through the success-trajectory filter and replay them as single-turn revision prompts, where standard single-turn credit assignment applies cleanly.

F.2 Wall-Clock Efficiency

Beyond avoiding credit-assignment pitfalls, our offline scheme is substantially faster than online multi-turn training. Multi-turn RL pays a length-M serial sequential-revision cost per prompt at every gradient step, multiplied by the G rollouts needed to estimate the advantage. Our method pays the length-K serial cost *only once* per prompt at the offline stage, and during the online RL stage performs only single-turn sampling on the extracted prompts, which can be parallelized across N Best-of-N samples. Single-turn parallel sampling is much cheaper per gradient step than M-turn serial generation across G rollouts, since serial generation does not benefit from the parallelism that modern inference engines provide for single-turn sampling.

Empirically, on $8\times$ NVIDIA H200 GPUs, three full iterations of our method completed in roughly 30 hours, while the equivalent online multi-turn baseline required over 48 hours on identical hardware (Table 5). With a more carefully optimized infrastructure, Stage I data generation and Stage II RL training could run concurrently (Stage II first trains on the original RL data while Stage I generates augmented data asynchronously), yielding further gains.

F.3 Sensitivity to the Confidence Threshold

We study the sensitivity of our test-time stopping rule to the confidence threshold c used in the `TailConfidence` stopping criterion (Section 4). Table 6 reports accuracy and the average number

Table 5: Per-iteration wall-clock breakdown of our method on $8\times$ NVIDIA H200 GPUs.

Iteration	Data Generation (Stage I)	RL Training (Stage II)	Total
Iter 1	3h 4m	7h 15m	10h 19m
Iter 2	2h 41m	7h 26m	10h 7m
Iter 3	3h 13m	6h 23m	9h 36m
Total	8h 58m	21h 4m	30h 2m

of revision rounds used at budget $K = 16$ across three math benchmarks. Higher thresholds enforce stricter stopping (more rounds before accepting a response), and accuracy increases monotonically with the threshold over the range we tested. The number of rounds also grows with the threshold, reflecting the cost-quality trade-off.

Table 6: Sensitivity of test-time accuracy to the `TailConfidence` threshold c on Qwen-2.5-7B-RL-Augmentation at budget $K = 16$. "Rounds" is the average number of revision rounds before stopping.

Threshold c	**AIME25**		**AIME24**		**MATH500**	
	Acc. (%)	Rounds	Acc. (%)	Rounds	Acc. (%)	Rounds
0.4	16.56	10.22	21.15	9.04	77.80	3.99
0.5	16.88	11.91	21.56	10.06	79.17	4.46
0.6	17.50	13.31	22.08	12.19	79.10	5.64
0.7	18.23	15.45	23.65	14.97	81.58	8.76

G Additional Experiments and Ablations

We additionally evaluate our method using a strong reasoning-oriented base model, DeepSeek-R1-Distill-7B (DeepSeek-AI et al., 2025), and compare against both the original base model and an RL baseline. Due to the substantially longer responses produced by this model (approximately 9k tokens on average, compared to $\sim$ 2k tokens for Qwen-7B), multi-turn training becomes significantly more expensive, which limits the scale of these experiments.

As shown in Table 7, our RL-Augmented model consistently improves test-case-based performance over both the base model and the RL baseline. We expect larger gains with increased data scale and longer training.

H Circle Packing Implementation Details

This section gives the implementation details behind the circle packing results in Figure 5. Circle packing maximizes the sum of radii of n non-overlapping circles inscribed in the unit square, with verifier feedback given by a non-overlap and inscription check together with the realized objective value. We instantiate REVES on this benchmark by directly optimizing $J_{\phi_{\mathrm{SR}}}$ in the recovery form of Lemma 4.1: each visited state z_t corresponds to a partially-improved candidate configuration, the policy proposes a refined configuration $y_t \sim \pi_\theta(\cdot \mid z_t)$, and the verifier returns $r^\star(x, y_t) = 1$ iff y_t improves on the running best while remaining feasible.

Table 7: Performance on LiveCodeBench under different evaluation periods.

Method	**LiveCodeBench (01/01/25–05/01/25)**		**LiveCodeBench (08/01/24–01/01/25)**	
	OneShot	**TestCases-32**	**OneShot**	**TestCases-32**
Base (Distill-7B)	34.25	44.51	38.24	52.21
Base + RL	**36.63**	43.96	38.79	53.19
Base + RL-Aug	35.53	**46.33**	**40.63**	**54.04**

Reward shaping. The raw score of a candidate is the sum of radii itself (a continuous quantity), which clusters in a narrow band near the optimum (≈ 2.63 for the $n = 26$ instance we evaluate) and differs across candidates only at the third or fourth decimal. Used directly, this signal is too compressed to drive learning. We therefore shape it in two steps. First, define the raw score $s(y)$ as

$$s(y) = \begin{cases} \text{(sum of radii of } y\text{)}, & y \text{ is feasible}, \\ \text{(sum of radii)} - \text{(constraint-violation amount)}, & y \text{ is infeasible and (sum of radii)} \leq 2.64, \\ -1, & y \text{ is infeasible and (sum of radii)} > 2.64, \end{cases}$$

i.e., minor violations are absorbed as a soft penalty, while implausibly high sums coupled with infeasibility are flagged as cheating and assigned $s = -1$. Second, we amplify by squaring the offset from 2.5: $R(s) := (s - 2.5)^2$. Since feasible raw scores live above 2.5, this magnifies small absolute differences, e.g., a $\Delta s = 10^{-3}$ gain near $s = 2.6$ produces a $\Delta R \approx 2 \cdot 10^{-4}$, which is no longer dwarfed by gradient noise. The reward used for a revision y from parent state x is the resulting improvement,

$$U(y \mid x) := R(s(y)) - R(s(x)),$$

so the policy is rewarded for any move that improves on the parent and penalized for any move that worsens it.

Stage I: search rollouts and prompt-set construction. We run 4 sequential-revision rollouts under the current π_θ with budget $K = 100$, where each step proposes a refinement of the previous candidate configuration and the verifier returns the shaped reward $U(y \mid x)$ defined above. From the visited intermediate states across these rollouts, we filter to retain those whose subsequent revisions produce verified improvements, and assemble the resulting revision prompts into the augmented prompt set.

Stage II: RL training. We train Qwen3-4B with GRPO (DeepSeek-AI et al., 2025) on the prompt set from Stage I. Each gradient step samples 8 prompts; for each prompt we draw 64 rollouts to form the group used for the GRPO advantage estimate.

A representative solution. Figure 9 visualizes one of the configurations REVES discovers, and Listing 1 reproduces the Python program emitted by the policy that generates it: an initial grid placement followed by an SLSQP refinement against the non-overlap and inscription constraints.

Listing 1: Python program output by the REVES-trained Qwen3-4B policy that produces the configuration in Figure 9.

```
import numpy as np
from scipy.optimize import minimize


def run_packing():
    num_circles = 26
    r_initial = 0.08
    x_positions = np.linspace(r_initial, 1 - r_initial, 6)
    y_positions = np.linspace(r_initial, 1 - r_initial, 5)
    positions = [(x, y) for x in x_positions for y in y_positions
        ][:25]
    positions.append((0.3, 0.3))
    positions = np.unique(positions, axis=0)
    if len(positions) < 26:
        positions = np.append(positions, [(0.5, 0.5)] * (26 - len(
            positions)), axis=0)

    radii = np.full(num_circles, r_initial)

    variables = np.zeros(num_circles * 3)
    for i in range(num_circles):
        variables[2 * i] = positions[i][0]
        variables[2 * i + 1] = positions[i][1]
    for i in range(num_circles):
        variables[2 * num_circles + i] = radii[i]

```

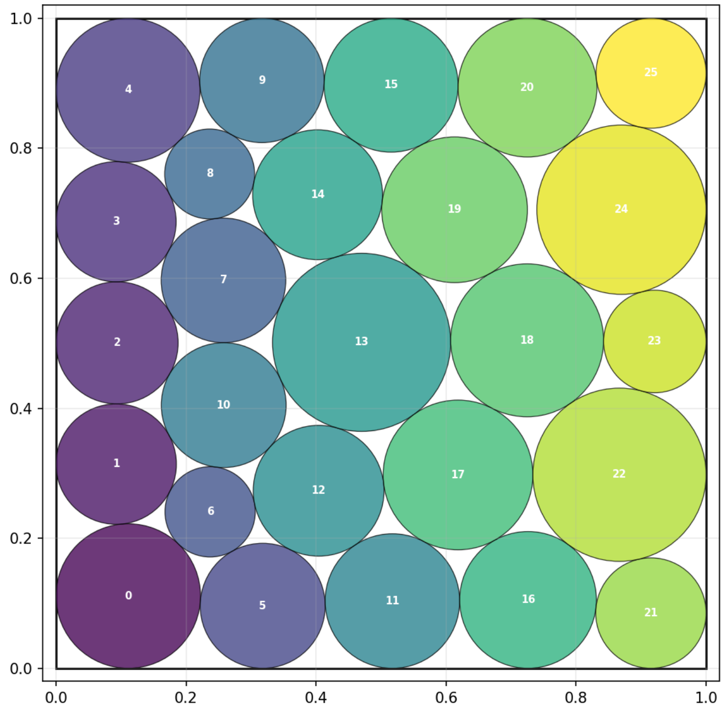


Figure 9: A circle packing configuration found by REVES on the $n = 26$ instance, achieving sum-of-radii $= 2.635983$.

```
    def objective(vars):
        return -np.sum(vars[num_circles * 2:])

    constraints = []
    for i in range(num_circles):
        def con1(vars, i=i):
            centers = vars[:num_circles * 2].reshape(num_circles, 2)
            r = vars[2 * num_circles + i]
            return centers[i, 0] - r
        constraints.append({'type': 'ineq', 'fun': con1})

        def con2(vars, i=i):
            centers = vars[:num_circles * 2].reshape(num_circles, 2)
            r = vars[2 * num_circles + i]
            return 1 - centers[i, 0] - r
        constraints.append({'type': 'ineq', 'fun': con2})

        def con3(vars, i=i):
            centers = vars[:num_circles * 2].reshape(num_circles, 2)
            r = vars[2 * num_circles + i]
            return centers[i, 1] - r
        constraints.append({'type': 'ineq', 'fun': con3})

        def con4(vars, i=i):
            centers = vars[:num_circles * 2].reshape(num_circles, 2)
            r = vars[2 * num_circles + i]
            return 1 - centers[i, 1] - r
        constraints.append({'type': 'ineq', 'fun': con4})

    for i in range(num_circles):
        def con5(vars, i=i):
            return vars[2 * num_circles + i]
```

```python
            constraints.append({'type': 'ineq', 'fun': con5})

    for i in range(num_circles):
        for j in range(i + 1, num_circles):
            def con6(vars, i=i, j=j):
                centers = vars[:num_circles * 2].reshape(num_circles,
                    2)
                r_i = vars[2 * num_circles + i]
                r_j = vars[2 * num_circles + j]
                dx = centers[i, 0] - centers[j, 0]
                dy = centers[i, 1] - centers[j, 1]
                return dx**2 + dy**2 - (r_i + r_j)**2
            constraints.append({'type': 'ineq', 'fun': con6})

    result = minimize(
        objective,
        variables,
        constraints=constraints,
        method='SLSQP',
        options={'maxiter': 2000000, 'ftol': 1e-10, 'disp': False}
    )

    centers = result.x[:num_circles * 2].reshape(num_circles, 2)
    radii = result.x[num_circles * 2:]
    return centers, radii, float(np.sum(radii))
```

Prompt Template for Code Revision

Problem

You will be given a problem specification and are asked to generate a correct Python program that satisfies the requirements and passes all tests. The task involves extracting two integers and an operator ("gains" or "loses") from a natural language string and returning the result of the corresponding calculation. . . .

Previous Answer

To address the issue in the provided code, there are a few key points ...

```python
class Solution:
    def calculate(self, string: str) -> int:
        words = string.split()
        ...
        joined_string = "".join(words)
        result = eval(joined_string)
        return result
```

. . .

Verifiable Feedback:
The previous answer is incorrect.

Instruction:
Please repair the code accordingly. Enclose your fixed implementation within the specified Python code delimiters. Do not assume access to the correct implementation.

Figure 10: Prompt template for the code revision stage with binary correctness feedback. Partial incorrect code is shown for illustration, with remaining content omitted for brevity.

Prompt Template for Code Verification

Problem

You will be given a problem specification and are asked to generate a correct Python program that satisfies the requirements and passes all tests. The task involves extracting two integers and an operator ("gains" or "loses") from a natural language string and returning the result of the corresponding calculation. . . .

Candidate Answer:

To address the issue in the provided code, there are a few key points ...

```
class Solution:
    def calculate(self, string: str) -> int:
        words = string.split()
        ...
        joined_string = "".join(words)
        result = eval(joined_string)
        return result
```

. . .

Instruction:

Please verify whether the candidate answer is correct. At the end, output the final line exactly as:
`VERDICT: \boxed{True/False}`

Figure 11: Prompt template for the code verification stage. The verifier evaluates a candidate solution and outputs a binary correctness verdict.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction reflect the paper's contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: See Appendix B for discussion on limitations.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: See Appendix C for the full set of assumptions and proofs.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See Appendix E for the implementation details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: I include the code in the supplemental material, data is publicly available.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See Appendix E and supplemental material for the implementation details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We run the experiments multiple times and report the mean.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Appendix E for the implementation details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research in the manuscript is compliant with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix B discusses this verifier requirement and natural future directions toward scientific-discovery applications.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We don't release any data or models that have a high risk for misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We credit the creators of the assets used in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.