# REVES: REVISION AND VERIFICATION-AUGMENTED TRAINING FOR TEST-TIME SCALING

**Anonymous authors**
Paper under double-blind review

## 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 the model can learn from correcting. 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. In a tractable surrogate where the exact gradient is computable, we prove that this per-state estimator has lower mean-squared error than multi-turn trajectory credit exactly below a sample-budget crossover that grows with problem difficulty and revision horizon. 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, and the advantage over SCoRe trained on the same data grows with the revision budget. Beyond coding, our approach matches the previously reported SOTA result on circle packing while using the smallest base model (4B) and far fewer rollouts than the much larger evolutionary search systems. 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), and evolutionary refinement (Lee et al., 2025b). 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 every revision-using TTS algorithm eventually calls the policy on revise-from-prior-attempt inputs, so gains on SR can carry over to the rest of the family (we test this directly), and because SR is a strong TTS algorithm in its own right, matching or surpassing tree search and evolutionary refinement on planning tasks (Appendix C), while being the simplest to train against.

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 deployment. The natural fix is multi-turn GRPO on revision rollouts, but it computes one group-



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

normalized advantage $A_m$ per trajectory and shares that scalar across all turns. In a wrong–wrong–correct rollout with $A_m > 0$, the two wrong intermediates are reinforced along with the final answer. At practical group sizes, this shared cross-turn credit is a noisy signal for the behavior we care about.

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 (Eq. (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, avoiding coarse cross-turn credit and the cost of long-horizon online sampling. We make the comparison with multi-turn RL precise in a finite-state surrogate where the exact gradient is computable: REVES's per-state estimator drops the visitation-flow term of the gradient but has structurally lower variance, and it has lower mean-squared error exactly when the per-state sample budget is below a crossover $M^\star$ that grows with problem difficulty and revision horizon (Theorem 4.1).

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 large improvements under ground-truth stopping and gains on AIME under 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 (MCTS, AB-MCTS variants, Mind Evolution).

Our contributions are threefold:

1. **Conceptual.** We formulate test-time scaling as a meta-RL problem in which the inference algorithm induces both the objective and the distribution of prompts the policy must handle, and show that the single-shot and sequential-revision objectives genuinely differ.
2. **Methodological.** We write $J_{\phi_{\mathrm{SR}}}$ exactly as a sum of per-state one-step recovery probabilities and propose REVES, a two-stage framework that recasts intermediate near-miss answers as decoupled revision and verification prompts trained with single-turn RL. In a tractable surrogate we prove when this per-state estimator has lower mean-squared error than multi-turn trajectory credit (Theorem 4.1).
3. **Empirical.** REVES delivers consistent test-time gains across math, coding, out-of-distribution puzzles, and circle packing, and improves every revision-using TTS algorithm we evaluated.

## 2 Related Work

**Test-time scaling and inference-aware post-training.** A growing family of test-time scaling algorithms spends extra compute at inference: parallel sampling with a selector (Wang et al., 2023; Chow et al., 2025), sequential revision with feedback (Madaan et al., 2023; Shinn et al., 2023), tree search (Inoue et al., 2025), and evolutionary refinement (Lee et al., 2025b). A parallel line of work aligns post-training with the intended inference strategy, so far mostly for parallel sampling, by optimizing pass@$k$ (Chen et al., 2025; Walder & Karkhanis, 2025; Tang et al., 2025). We target the *revision-using* family, whose calls condition on a previous attempt and its feedback, and therefore change both the objective and the distribution of prompts the policy must handle.

**Meta-RL and self-correction training.** Our formulation treats the inference algorithm as inducing a decision process in which the same policy is called repeatedly; this is the meta-RL view also taken by MRT (Qu et al., 2025) and, for agents, LaMer (Jiang et al., 2026b), which optimize cross-episode returns with multi-turn policy gradients. Methods that train self-correction directly, SCoRe (Kumar et al., 2024), ExIt (Jiang et al., 2026a), and related work (Qu et al., 2024; Jiang et al., 2025; Yuan & Xie, 2025), share our goal; REVES differs in deriving its per-state training task from the first-success objective of the revision-using controller and in analyzing the resulting estimator

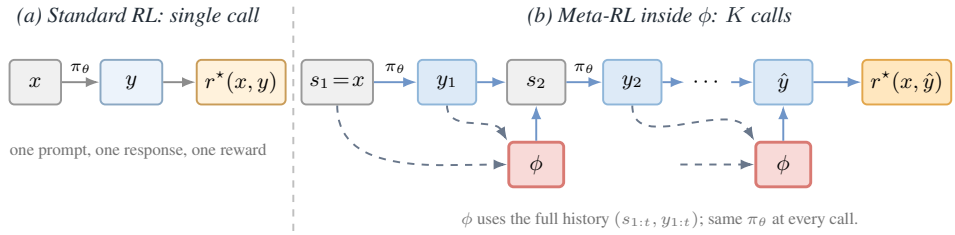



Figure 1: **(a)** Standard RL: policy $\pi_\theta$ maps problem $x$ to response $y$ with reward $r^\star(x, y)$. **(b)** A TTS algorithm $\phi$ invokes the same $\pi_\theta$ over up to $K$ steps, each prompt $s_t$ built from the prior history (dashed arrows), and extracts a final response $\hat{y} =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

against trajectory-level credit (Section 4.2). A detailed comparison with each of these methods is in Appendix L.

## 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 $\pi_\theta$ in RL parameterized by $\theta$. At deployment, an LLM rarely operates by a single call to $\pi_\theta$; instead a *test-time scaling (TTS) algorithm* $\phi$ 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 B.

**General formulation: the TTS-induced decision process.** Pairing any TTS algorithm $\phi$ with $\pi_\theta$ 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 $\phi$ constructs the next prompt via a transition kernel $P_\phi(s_{t+1} \mid s_{1:t}, y_{1:t})$. The initial state is $s_1 = x$. The policy itself is Markov in the current prompt $s_t$; the history dependence sits in the kernel, since $\phi$ 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 $\phi$ chooses the prior step $i_t$: SR always revises the most recent response ($i_t = t$), so the induced process is a standard MDP, whereas tree search such as AB-MCTS may branch from any earlier response, making the kernel history-dependent. The procedure runs until a random stopping time $\tau := K \wedge \min\{t : r^\star(x, y_t) = 1\}$ and outputs $\hat{y} = y_\tau$: it *early-stops* at the first verified-correct response or *exhausts the budget* $K$. Running $\pi_\theta$ inside $\phi$ yields the TTS-induced objective $J_\phi(\theta) = \mathbb{E}_{x \sim \mathcal{X},\, (s_{1:\tau}, y_{1:\tau}) \sim (\pi_\theta, \phi)}[r^\star(x, y_\tau)]$, to be contrasted with the standard single-shot objective $J_{\mathrm{OneShot}}(\theta) = \mathbb{E}_{x \sim \mathcal{X},\, y \sim \pi_\theta(\cdot|x)}[r^\star(x, y)]$, which is what RLHF (Ouyang et al., 2022), RLVR, and GRPO (DeepSeek-AI et al., 2025) optimize. Optimizing $J_\phi$ rather than $J_{\mathrm{OneShot}}$ is a *meta-RL* problem (Figure 1): the policy must adapt to the multi-step contexts that $\phi$ generates, not just answer the original question, and our goal is to improve $J_\phi$ for every $\phi \in \Phi_\mathcal{R}$.

**Sequential revision as the training controller.** Every algorithm in $\Phi_\mathcal{R}$ at some point invokes $\pi_\theta$ to revise a previous attempt, so raising the policy's one-step recovery on revision inputs that a deployment-time $\phi$ also visits should benefit $\phi$. We do not formalize this transfer; we test it directly by placing the trained policy inside other controllers (Table 3).

**Our training target $J_{\phi_{\mathrm{SR}}}$.** We pick $\phi_{\mathrm{SR}}$ and optimize $J_{\phi_{\mathrm{SR}}}(\theta) := \mathbb{E}_{x\sim\mathcal{X},\, y_{1:\tau}\sim(\pi_\theta,\phi_{\mathrm{SR}})}[r^\star(x, y_\tau)]$ for two reasons. First, $\phi_{\mathrm{SR}}$ is itself a strong TTS algorithm: on TravelPlanner (Xie et al., 2024) it matches or surpasses Mind Evolution (Lee et al., 2025b) and AB-MCTS (Inoue et al., 2025), and its Markov-revision form $\pi_\theta(\cdot \mid x, y_{\mathrm{prev}}, f)$ lies on the best cost-accuracy frontier (Appendix C). Second, it is the simplest member of $\Phi_{\mathcal{R}}$ to train against: one linear revision trajectory, no search tree, population, or value model, and it admits the exact recovery decomposition of Section 4.

**Optimizing $J_{\phi_{\mathrm{SR}}}$ is not optimizing single-shot pass@1.** Standard single-shot RLVR optimizes $J_{\mathrm{OneShot}}$, which does not in general determine $J_{\phi_{\mathrm{SR}}}$: policies with identical pass@1 can have different sequential-revision success (Appendix D). We therefore ask how to post-train $\pi_\theta$ to improve $J_{\phi_{\mathrm{SR}}}$; the next section presents our answer.

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

Section 3 identified $J_{\phi_{\mathrm{SR}}}$ as our target. We now derive REVES in three steps: decompose $J_{\phi_{\mathrm{SR}}}$ into per-state recovery probabilities (Section 4.1), compare their estimator with multi-turn trajectory credit in a tractable model (Section 4.2), and introduce the practical algorithm (Section 4.3).

## 4.1 What REVES optimizes: recovery at frozen states

The direct way to optimize $J_{\phi_{\mathrm{SR}}}$ is multi-turn GRPO on SR rollouts. For each prompt, it group-normalizes the rollouts' terminal rewards $R_{1:M}$ into trajectory advantages $A_{1:M}$ and assigns the same $A_m$ to every response in rollout $m$. Figure 2(a) illustrates the problem. In a wrong–wrong–correct rollout with $A_m > 0$, the two incorrect responses are reinforced along with the final correct one. Multi-turn RL can therefore increase the likelihood of an incorrect response simply because a later revision makes its trajectory successful.

**Rewriting the objective around recovery.** Let $\zeta = (z_1, y_1, \ldots, z_\tau, y_\tau) \sim (\pi_\theta, \phi_{\mathrm{SR}})$ be a stopped SR trajectory, with $z_t = (x, y_{t-1}, f_{t-1})$ the revision state before the $t$-th attempt. Define $V_\pi(z) := \mathbb{P}_{y'\sim\pi_\theta(\cdot|z)}(r^\star(x, y') = 1)$ as the probability that one fresh attempt from $z$ is correct, and let $\rho_\theta(z)$ be the expected number of visits to $z$ in an SR run. Because SR stops at the first success (proof in Appendix F), its trajectory objective can be rewritten exactly as

$$J_{\phi_{\mathrm{SR}}}(\theta) = \mathbb{E}_{x\sim\mathcal{X},\, \zeta\sim(\pi_\theta,\phi_{\mathrm{SR}})}\left[\sum_{t=1}^{\tau} V_\pi(z_t)\right] = \sum_z \rho_\theta(z)\, V_\pi(z). \tag{1}$$

Thus, $J_{\phi_{\mathrm{SR}}}$ is a visitation-weighted sum of one-step recovery probabilities. The contribution associated with a visited state is its *own* recovery probability, precisely the local signal obscured by trajectory-level credit.

**From the rewritten objective to REVES.** Equation (1) separates state visitation from per-state recovery. REVES freezes visitation under the current policy and optimizes recovery:

$$J^{\bar\rho}(\pi) := \sum_z \bar\rho(z)\, V_\pi(z), \qquad \bar\rho := \rho_{\theta_{\mathrm{old}}}, \tag{2}$$

This is a single-turn RL problem on the prompt set $\{z : \bar\rho(z) > 0\}$. We run SR under the current policy to collect states, then draw a fresh response group at each state and compute per-response GRPO advantages from the correctness rewards $r^\star(x, y')$ (Figure 2(b)). A wrong intermediate response is no longer trained as an action that inherits the eventual trajectory outcome; it becomes a state from which the policy learns to recover.

## 4.2 Gradient estimation as a bias–variance tradeoff

To isolate the statistical difference between REVES and multi-turn RL, we compare their gradient estimators in a controlled setting where the exact gradient and both mean-squared errors (MSEs) are computable. This lets us ask directly when the lower-variance local credit used by REVES outweighs the bias from freezing state visitation.

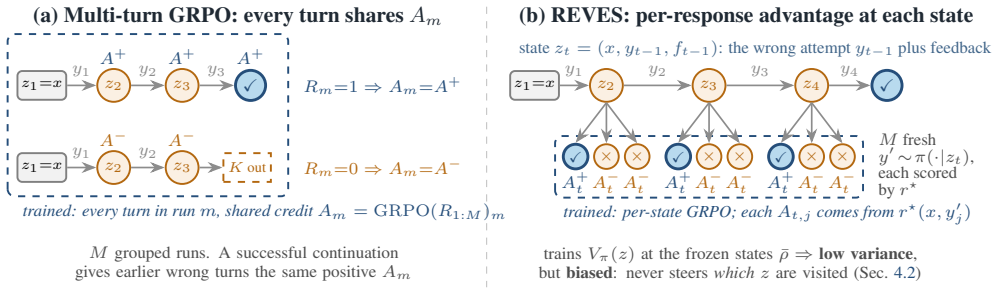



Figure 2: Shared trajectory advantage versus per-state advantages. An orange node represents an incorrect response $y_t$ and is labeled by the next revision state $z_{t+1} = (x, y_t, f_t)$ that it creates; a blue ✓ marks a correct response. Dashed navy boxes enclose the samples used for policy-gradient updates. **(a)** Multi-turn GRPO group-normalizes the $M$ terminal rewards $R_{1:M}$ into trajectory advantages $A_{1:M}$, then assigns the same $A_m$ to every response in rollout $m$. Thus, a positive $A_m$ reinforces incorrect responses whenever a later revision makes the rollout successful. **(b)** REVES uses the rollout only to collect states. At each state, it draws a fresh response group and computes per-response GRPO advantages from the individual correctness rewards $r^\star(x, y')$.

We use a finite-state Markov revision model with transient states $z_1, \ldots, z_N$ and an absorbing success state. At each $z_i$, the policy has an independent logit vector $\theta_i \in \mathbb{R}^{N+1}$ whose softmax defines one "succeed" action and $N$ "move-to-$z_j$" actions. Hence the same policy row determines both the one-step recovery probability $P_i = \pi_i(\text{succeed})$ and the transition probabilities $Q_{ij} = \pi_i(\text{move-to-}z_j)$. Starting from $z_1$, the process stops upon success or after $K$ attempts, and its exact success objective is $J = \sum_i \rho_i P_i$. We compare the unnormalized REINFORCE counterparts of the two methods: given sampling budget $M$, $\hat{g}^{\mathrm{MT}}$ averages $M$ trajectory-level samples, while $\hat{g}^{\mathrm{RE}}$ uses one trajectory to select visited states and draws $M$ fresh one-step samples at each selected state. This model makes the exact gradient and both MSEs computable; full definitions are in Appendix N. Practical GRPO changes the reward normalization but preserves the distinction between trajectory-shared and per-state credit. Here $M$ is a sampling parameter—the per-state group size for REVES—rather than a claim of equal token cost.

**Theorem 4.1** (Bias–variance crossover, informal; formal statement and proof in Appendix N)**.** *In the toy model, $\hat{g}^{\mathrm{MT}}$ is unbiased, $\hat{g}^{\mathrm{RE}}$ is biased, and $\mathrm{MSE}(\hat{g}^{\mathrm{MT}}) = a_{\mathrm{MT}}/M$ while $\mathrm{MSE}(\hat{g}^{\mathrm{RE}}) = F + c_{\mathrm{RE}}/M$, where $a_{\mathrm{MT}}$ is the variance of one multi-turn run, $c_{\mathrm{RE}}$ is the variance of one fresh branch, and $F > 0$ is REVES's bias floor. When $a_{\mathrm{MT}} > c_{\mathrm{RE}}$, the two curves cross once at $M^\star = (a_{\mathrm{MT}} - c_{\mathrm{RE}})/F$: REVES has lower MSE for $M < M^\star$, and multi-turn RL has lower MSE for $M > M^\star$. If $a_{\mathrm{MT}} \le c_{\mathrm{RE}}$, multi-turn RL has lower MSE for every $M$.*

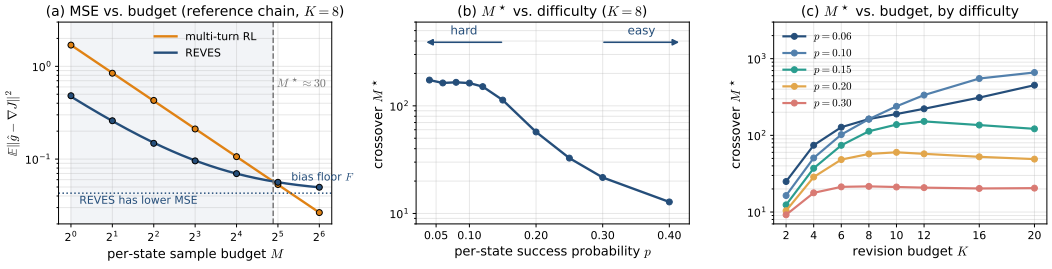



Figure 3: The lower-MSE region in the toy model. **(a)** Gradient MSE of multi-turn RL (orange) and REVES (blue) as the per-state sample budget $M$ grows; markers are direct Monte-Carlo estimates. REVES has lower MSE for $M < M^\star$, but its curve approaches a bias floor while multi-turn RL continues to improve. **(b–c)** The boundary $M^\star$ moves with problem difficulty and revision budget: harder problems and longer budgets expand the range of $M$ in which REVES has lower MSE, while the effect of budget saturates on easy problems.

The theorem identifies the sample budget as the key variable in the bias–variance tradeoff. REVES accepts a fixed bias in exchange for lower variance, so it has lower MSE when samples are limited. Multi-turn RL is unbiased, however, and eventually becomes better as averaging removes its trajectory-level variance. The crossover $M^\star$ separates these two regimes: REVES is preferable for $M < M^\star$, and multi-turn RL for $M > M^\star$ (Figure 3(a)).

The size of this REVES-favorable regime depends on the task. When per-state success is lower, trajectories are longer and trajectory-level credit is noisier, pushing $M^\star$ upward. Increasing the revision budget has the same effect on hard problems; on easy problems, trajectories usually terminate early, so $M^\star$ saturates. Thus, REVES's variance reduction is most useful on difficult, long-horizon revision problems with finite sampling budgets (Figure 3(b,c)).

### 4.3 From the recovery target to a practical algorithm

An ideal on-policy implementation of (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 Appendix I.2.

*(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 $\pi_\theta$ improves. Appendix I.3 compares this filter against unfiltered and difficulty-matched alternatives.

**Two-stage realization.** **Stage I (Data augmentation).** For each input $x$, run SR under the current $\pi_\theta$: $y_1 \sim \pi_\theta(\cdot \mid x)$ and $y_t \sim \pi_\theta(\cdot \mid x, y_{t-1}, \texttt{Feedback}(y_{t-1}))$ for $t = 2, \ldots, K$, terminating at 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* $\{\texttt{Revision Prompt}(x, y_i)\}_{i=1}^{\ell-1}$ from the incorrect attempts, where each prompt asks the model to revise the previous response, and (ii) *verification data* $\{\texttt{Verification Prompt}(x, y_i)\}_{i=1}^{\ell}$ labeled by $r^\star$, where each prompt asks the model to judge whether $y_i$ is correct; the final correct response $y_\ell$ is never presented as incorrect (pseudocode in Appendix M). Templates are in Appendix G. **Stage II (Single-turn RL).** Train $\pi_\theta$ 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 I.6 reports the wall-clock comparison.

**Test-time stopping rule.** At test time the oracle stopping time $\tau$ 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 $\hat{\tau} := K \wedge \min\left\{t \geq 3 : c_t / \sum_{j=1}^{t} c_j > c\right\}$

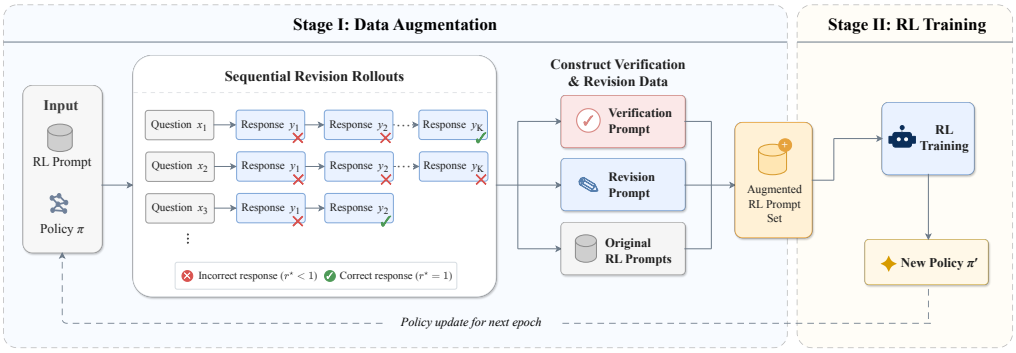



Figure 4: Overview of the proposed RL-based data augmentation framework. In Stage I, the current policy performs sequential revision; only trajectories that succeed within budget $K$ are retained, and their intermediate states yield revision and verification prompts that are merged with the original 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.

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, matched to REVES on sampled responses per update and realized generated tokens; Appendix I.1), **PAG** (Jiang et al., 2025) (multi-turn with self-verification via a designed reward), **SCoRe** (Kumar et al., 2024) (trained with its two-stage recipe on the same model, data, and gradient-step budget), and a **pass@$k$-trained** policy (PKPO (Walder & Karkhanis, 2025), $k$=8). Full details in Appendix H.

### 5.1 Main Results

**Consistent improvements on coding tasks.** Table 1 demonstrates the benefits of REVES 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 B for an example), REVES 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 J.

**Improved sequential revision capability for mathematical reasoning.** Table 2 shows that REVES 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

Table 1: Sequential-revision test-time scaling performance on coding benchmarks (LiveCodeBench (LCB), CodeContest). The best result in each column is shown in bold, and the second-best is underlined. TC-$B$ 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.

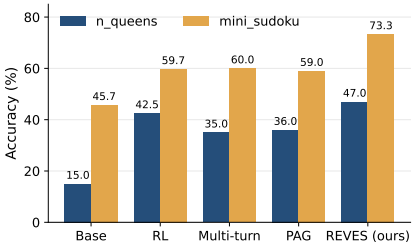
| Model | LCB (Aug 24 – Jan 25) | | | LCB (Jan 25 – May 25) | | CodeContest | | |
|---|---|---|---|---|---|---|---|---|
| | OneShot | TC-32 | TC-4 | OneShot | TC-32 | OneShot | TC-32 | TC-4 |
| Qwen-2.5-7B | 4.8 | 6.1 | 6.1 | 6.0 | 18.9 | 0.4 | 5.7 | 5.9 |
| Qwen-2.5-7B-RL | 18.0 | 23.0 | 20.8 | 20.0 | 24.2 | **7.1** | 11.1 | 7.5 |
| Qwen-2.5-7B-Multi-turn | 19.4 | 25.5 | 21.3 | **20.7** | 27.5 | 6.9 | 14.3 | 8.1 |
| Qwen-2.5-7B-PAG | **19.5** | 25.7 | 23.2 | 19.8 | 26.4 | 6.5 | 15.0 | 9.1 |
| Qwen-2.5-7B-REVES | 18.9 | **29.5** | **23.8** | 20.5 | **30.0** | **7.1** | **16.6** | **11.7** |
| Qwen-2.5-3B | 5.9 | 15.8 | 7.2 | 7.9 | 18.3 | 1.6 | 6.3 | 2.4 |
| Qwen-2.5-3B-RL | 9.2 | 12.0 | 11.5 | 14.1 | 17.8 | 3.0 | 4.2 | 3.6 |
| Qwen-2.5-3B-Multi-turn | **9.9** | 15.7 | **12.1** | 13.2 | 19.2 | 3.2 | 7.9 | 4.4 |
| Qwen-2.5-3B-PAG | 8.1 | 14.7 | 11.2 | 13.4 | 18.5 | **3.6** | 5.7 | 4.9 |
| Qwen-2.5-3B-REVES | 9.6 | **17.7** | 11.0 | **14.3** | **21.6** | 2.2 | **8.5** | **5.5** |
| Qwen3-4B (base) | 22.6 | 27.8 | 26.8 | 24.7 | 24.7 | 6.1 | 10.9 | 9.7 |
| Qwen3-4B-RL | 32.2 | 41.7 | 42.5 | 29.1 | 35.4 | 16.1 | 30.9 | 30.9 |
| Qwen3-4B-Multi-turn | 29.2 | 49.5 | 44.9 | 25.3 | 40.9 | 10.6 | 34.9 | **34.2** |
| Qwen3-4B-PAG | 29.8 | 43.6 | 40.4 | 26.7 | 37.1 | 13.6 | 32.4 | 25.8 |
| Qwen3-4B-REVES | **32.7** | **50.9** | **46.3** | **29.7** | **42.0** | **16.4** | **37.9** | 30.9 |

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, on AIME24/25, indicating that the learned capability transfers beyond the oracle setting; on MATH500, where base accuracy is already high, SelfConf-4 is within noise of RL. SCoRe, trained on the same data and step budget, stays close to RL under both OneShot and Oracle-32, whereas REVES is far ahead of it at Oracle-32 on all three benchmarks despite similar or lower OneShot accuracy, so the gain comes from revision rather than from a stronger first attempt. The gap over SCoRe also widens monotonically as the revision budget grows (Appendix I.4), the empirical counterpart of the horizon effect in Section 4.2.

**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 of much larger evolutionary search systems, AlphaEvolve (Novikov et al., 2025; Georgiev et al., 2025), ShinkaEvolve (Lange et al., 2025), ThetaEvolve (Wang et al., 2025), and TTT-Discover (Yuksekgonul et al., 2026), which build on Gemini-2.0 Pro/Flash or Qwen3-8B (Figure 5(a)). Full implementation details are in Appendix K.

| Method | Model | Sum of radii, $n$=26 (↑) |
|---|---|---|
| AlphaEvolve | Gemini-2.0 Pro + Flash | 2.635862 |
| AlphaEvolve V2 | Gemini-2.0 Pro + Flash | **2.635983** |
| ShinkaEvolve | Ensemble | 2.635982 |
| ThetaEvolve | R1-Qwen3-8B | **2.635983** |
| TTT-Discover | Qwen3-8B | **2.635983** |
| Ours (REVES) | **Qwen3-4B** | **2.635983** |

(a) Circle packing




(b) Out-of-distribution puzzles

Figure 5: **(a)** Circle packing (sum of radii at $n$=26, ↑): REVES on Qwen3-4B matches the best previously reported result, reaching the optimal 2.635983 with the *smallest* base model and fewer rollouts. **(b)** Out-of-distribution puzzle benchmarks (n_queens and mini_sudoku) under verifier-based stopping, using checkpoints trained only on math and code.

Table 2: Sequential-revision test-time scaling performance on math benchmarks (AIME24/25, MATH500). **1-shot** is OneShot; **O-**$B$ (Oracle) and **SC-**$B$ (SelfConf) are sequential-revision (SR) test-time scaling with budget $B$, using the ground-truth answer as the oracle stopping rule and the model's own confidence as a surrogate stopping rule, respectively. † denotes evaluation using the authors' original codebase and protocol.

| Model | AIME 24 | | | | AIME 25 | | | | MATH 500 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1-shot | O-32 | O-4 | SC-4 | 1-shot | O-32 | O-4 | SC-4 | 1-shot | O-32 | O-4 | SC-4 |
| Qwen-2.5-7B | 2.0 | 14.0 | 4.8 | 2.3 | 0.3 | 6.3 | 1.3 | 0.6 | 23.8 | 50.1 | 29.0 | 26.1 |
| Qwen-2.5-7B-RL | **17.9** | 33.5 | 21.4 | 18.7 | **10.4** | 22.9 | 13.4 | 12.5 | 76.1 | 85.9 | 79.4 | 78.6 |
| Qwen-2.5-7B-Multi-turn | 14.6 | 30.3 | 20.0 | 13.8† | 7.2 | 25.3 | 11.8 | 9.8† | 75.9 | 87.1 | 80.1 | 78.0† |
| Qwen-2.5-7B-PAG | 15.3 | 31.1 | 19.6 | 13.0† | 7.5 | 26.8 | 12.8 | 9.4† | 76.1 | 86.7 | 80.8 | 77.2† |
| Qwen-2.5-7B-SCoRe | 17.1 | 34.8 | 23.0 | **20.0** | 8.3 | 27.6 | 14.2 | 10.2 | 75.7 | 88.6 | 78.8 | **78.7** |
| Qwen-2.5-7B-REVES | 14.9 | **45.7** | **25.9** | 19.6 | 9.3 | **40.5** | **22.3** | **15.1** | **76.8** | **94.7** | **85.6** | 76.2 |
| Qwen-2.5-3B | 1.2 | 12.9 | 3.4 | 1.9 | 0.5 | 7.9 | 1.3 | 0.6 | 30.5 | 67.0 | 39.3 | 38.4 |
| Qwen-2.5-3B-RL | 6.4 | 16.9 | 8.1 | 7.4 | 1.8 | 9.0 | 3.9 | 2.4 | 64.9 | 80.5 | 69.6 | 65.6 |
| Qwen-2.5-3B-Multi-turn | **7.6** | 20.5 | 12.0 | 8.0† | 2.6 | 12.7 | 5.2 | 2.3† | **65.3** | 81.2 | 72.7 | **67.0**† |
| Qwen-2.5-3B-PAG | 6.3 | 14.8 | 7.9 | 5.3† | **3.4** | 11.4 | 4.7 | 4.1† | 64.0 | 78.8 | 69.9 | 65.6† |
| Qwen-2.5-3B-REVES | 6.2 | **33.7** | **14.7** | **8.2** | 2.8 | **24.4** | **7.2** | **4.2** | 64.5 | **89.9** | **77.8** | 66.0 |

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

**$J_{\phi_{\text{SR}}}$-trained policies also improve other TTS algorithms.** Table 3 shows that models trained with REVES consistently improve the performance of a wide range of test-time algorithms that rely on sequential revision, including 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. The pass@$k$-trained policy is strongest on best-of-$N$ (its own objective) and SCoRe on Mind Evolution, but neither reaches REVES on the controllers whose calls condition on prior attempts (SeqRev, MCTS, and the AB-MCTS variants).

Table 3: Comparison of model performance on different test-time strategies. Best results in each column are in bold; second-best are underlined. **1-shot** is OneShot; **MindEvo** is Mind Evolution; **ABM-A($\beta$)**, **ABM-A(G)**, and **ABM-M** are the AB-MCTS-A (Beta), AB-MCTS-A (Gaussian), and AB-MCTS-M variants.

| Model | 1-shot | SeqRev | BoN | MCTS | MindEvo | ABM-A($\beta$) | ABM-A(G) | ABM-M |
|---|---|---|---|---|---|---|---|---|
| Qwen-2.5-7B-RL | 18.0 | 23.0 | 27.8 | 22.7 | 25.7 | 26.0 | 27.3 | 26.2 |
| Qwen-2.5-7B-Multi-turn | 19.4 | 25.5 | 28.7 | 26.1 | 27.0 | 28.6 | 29.0 | 27.8 |
| Qwen-2.5-7B-PAG | **19.5** | 25.7 | **30.4** | 26.4 | 27.7 | 29.7 | 28.1 | 28.7 |
| Qwen-2.5-7B-SCoRe | 18.4 | 27.4 | 30.2 | 26.1 | **31.3** | 29.4 | 28.7 | 29.0 |
| Qwen-2.5-7B-pass@$k$ | 16.8 | 27.4 | **30.4** | 28.3 | 29.4 | 29.8 | 29.4 | 28.3 |
| Qwen-2.5-7B-REVES | 18.9 | **29.5** | 29.3 | **28.9** | 29.0 | **31.0** | **30.0** | **30.4** |

### 5.2 Ablation Study

Appendices I.2, I.3, and I.5 ablate prompt composition, state selection, and data refresh. Continual re-augmentation consistently outperforms one-shot augmentation on AIME24 and MATH500, while longer Stage-I trajectories improve accuracy at additional generation cost. Revision prompts drive the improvement in sequential revision, whereas verification prompts improve confidence calibration. Success-filtered states outperform both unfiltered training and difficulty-matched replay; detailed feedback offers no consistent gain over binary feedback. Overall, the ablations validate REVES's three data-design choices.

### Reproducibility statement

Training data, base models, decoding parameters, budgets, and stopping rules are specified in Section 5 and Appendix H; the sequential-revision protocol, prompt templates, and worked examples are in Appendices B and G; the proof of (1) is in Appendix F; the surrogate model of Section 4.2 is fully specified in Appendices N and P, and the scripts that reproduce every number and figure in that section are in the repository below. Code for data augmentation, training, and the evaluation harness is available at the anonymous repository https://anonymous.4open.science/r/REVE-0D68; augmented prompt sets and checkpoints will be released upon publication.

### AI use statement

Generative AI tools were used for literature retrieval and discovery, research ideation, drafting and polishing portions of the manuscript, and writing auxiliary analysis and plotting scripts. All AI-assisted text was reviewed and revised by the authors, all scripts were checked and run by the authors, and the authors take responsibility for the final content of this work, including all claims and artifacts.

## References

Zhipeng Chen, Xiaobo Qin, Youbin Wu, Yue Ling, Qinghao Ye, Wayne Xin Zhao, and Guang Shi. Pass@k training for adaptively balancing exploration and exploitation of large reasoning models, 2025. URL https://arxiv.org/abs/2508.10751.

Yinlam Chow, Guy Tennenholtz, Izzeddin Gur, Vincent Zhuang, Bo Dai, Sridhar Thiagarajan, Craig Boutilier, Rishabh Agarwal, Aviral Kumar, and Aleksandra Faust. Inference-aware fine-tuning for best-of-n sampling in large language models, 2025. URL https://arxiv.org/abs/2412.15287.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence, 2025. URL https://arxiv.org/abs/2508.15260.

Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. Skywork open reasoner 1 technical report, 2025.

Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? scaling llm inference-time compute with adaptive branching tree search, 2025. URL https://arxiv.org/abs/2503.04412.

Arnav Kumar Jain, Gonzalo Gonzalez-Pumariega, Wayne Chen, Alexander M Rush, Wenting Zhao, and Sanjiban Choudhury. Multi-turn code generation through single-step rewards, 2025. URL https://arxiv.org/abs/2502.20380.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Minqi Jiang, Andrei Lupu, and Yoram Bachrach. Bootstrapping task spaces for self-improvement. *Transactions on Machine Learning Research*, 2026a. arXiv:2509.04575.

Yuhua Jiang, Yuwen Xiong, Yufeng Yuan, Chao Xin, Wenyuan Xu, Yu Yue, Qianchuan Zhao, and Lin Yan. Pag: Multi-turn reinforced llm self-correction with policy as generative verifier, 2025. URL https://arxiv.org/abs/2506.10406.

Yulun Jiang, Liangze Jiang, Damien Teney, Michael Moor, and Maria Brbić. Meta-rl induces exploration in language agents. In *International Conference on Learning Representations (ICLR)*, 2026b. arXiv:2512.16848.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning, 2024. URL https://arxiv.org/abs/2409.12917.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

Hyunseok Lee, Seunghyuk Oh, Jaehyung Kim, Jinwoo Shin, and Jihoon Tack. Revise: Learning to refine at test-time via intrinsic self-verification, 2025a. URL https://arxiv.org/abs/2502.14565.

Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. Evolving deeper llm thinking, 2025b. URL https://arxiv.org/abs/2501.09891.

Ang Li, Yifei Wang, Zhihang Yuan, Stefanie Jegelka, and Yisen Wang. Lanpo: Bootstrapping language and numerical feedback for reinforcement learning in llms, 2025a. URL https://arxiv.org/abs/2510.16552.

Yanyang Li, Michael Lyu, and Liwei Wang. Learning to reason from feedback at test-time, 2025b. URL https://arxiv.org/abs/2502.15771.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://www.science.org/doi/abs/10.1126/science.abq1158.

Xiao Liang, Zhong-Zhi Li, Yeyun Gong, Yang Wang, Hengyuan Zhang, Yelong Shen, Ying Nian Wu, and Weizhu Chen. Sws: Self-aware weakness-driven problem synthesis in reinforcement learning for llm reasoning, 2025. URL https://arxiv.org/abs/2506.08989.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve, 2024. URL https://arxiv.org/abs/2407.18219.

Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-tuning. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2503.07572.

Yuxiao Qu, Amrith Setlur, Virginia Smith, Ruslan Salakhutdinov, and Aviral Kumar. Pope: Learning to reason on hard problems via privileged on-policy exploration, 2026. URL https://arxiv.org/abs/2601.18779.

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

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards, 2025. URL https://arxiv.org/abs/2505.24760.

Yunhao Tang, Kunhao Zheng, Gabriel Synnaeve, and Rémi Munos. Optimizing language models for inference time objectives using reinforcement learning, 2025. URL https://arxiv.org/abs/2503.19595.

Christian Walder and Deep Karkhanis. Pass@k policy optimization: Solving harder reinforcement learning problems, 2025. URL https://arxiv.org/abs/2505.15201.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, et al. Thetaevolve: Test-time learning on open problems. *arXiv preprint arXiv:2511.23473*, 2025.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents, 2024. URL https://arxiv.org/abs/2402.01622.

Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, Chi Jin, Tong Zhang, and Tianqi Liu. Building math agents with multi-turn iterative preference learning, 2025. URL https://arxiv.org/abs/2409.02392.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yurun Yuan and Tengyang Xie. Reinforce llm reasoning through multi-agent reflection. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2506.08379.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time, 2026. URL https://arxiv.org/abs/2601.16175.

Xiaoying Zhang, Yipeng Zhang, Hao Sun, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. Critique-grpo: Advancing llm reasoning with natural language and numerical feedback, 2026. URL https://arxiv.org/abs/2506.03106.

# A Conclusion

We show that standard single-shot RL objectives are misaligned with test-time sequential revision, and propose **REVES**, which pairs RL with continual, mistake-driven data augmentation so the model learns both to revise incorrect solutions and to identify errors. Augmentation must be continual, since static data becomes uninformative once mistakes disappear. Across math, coding, and out-of-distribution tasks, REVES improves test-time sequential-revision performance over the single-shot RL and multi-turn baselines under fixed budgets, and these gains carry over to other revision-using test-time algorithms. On circle packing, a Qwen3-4B trained with REVES matches the previously reported state-of-the-art achieved by much larger evolutionary search systems, using the smallest base model and far fewer rollouts. In a tractable surrogate, we further characterize when REVES's per-state recovery estimator has lower error than multi-turn trajectory credit: below a sample-budget crossover that grows with problem difficulty and revision horizon.

# B 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 B.1 and a coding task in Appendix B.2.

**Protocol 1** Sequential Revision Protocol

**Input:** revision budget $K$, question $x$, model $\pi$
Sample initial response $y_1 \sim \pi(\cdot \mid x)$
**for** $i = 2$ **to** $K$ **do**
  Generate revised response $y_i \sim \pi(\cdot \mid x, y_{i-1}, \mathsf{Feedback}(y_{i-1}))$
  **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**

## B.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 $90^\circ$ 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 $90^\circ$ clockwise rotation around the complex number $-4 - 5i$. First, it's useful to understand that a $90^\circ$ 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 $90^\circ$ 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 $i^2 = -1$:

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

$$\dots$$

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

**Input (Turn 2)**

Problem: A $90^\circ$ 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 $90^\circ$ clockwise rotation around the complex number $-4 - 5i$.
First, it's useful to understand that a $90^\circ$ 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 $90^\circ$ 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 $90^\circ$ 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 $90^\circ$ 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 $90^\circ$ clockwise rotation, the resulting complex number is $\boxed{-9 + 2i}$.

**Input (Turn 3)**

Problem: A $90^\circ$ 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 $90^\circ$ 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 $90^\circ$ 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 $90^\circ$ 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 $90^\circ$ 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 $90^\circ$ clockwise:** A $90^\circ$ 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 $90^\circ$ clockwise rotation is $\boxed{1 - 12i}$.

### B.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 $P = (P_1, P_2, \ldots, P_N)$ of $(1, 2, \ldots, N)$. You want to satisfy $P_i = i$ for all $i = 1, 2, \ldots, N$ 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 $i = 1, 2, \ldots, N$ 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 $(1, 2, \ldots, N)$. 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))
```

**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))
```

## C Sequential Revision as a Test-Time Scaling Algorithm

Figure 6 compares sequential revision with tree-search and evolutionary controllers on TravelPlanner (Xie et al., 2024) using DeepSeek-V3. SR matches or surpasses Mind Evolution and AB-MCTS (left), and the Markov-revision form that conditions only on the most recent attempt lies on the best cost–accuracy frontier (right); together with its simplicity, this motivates SR as the training controller (Section 3).

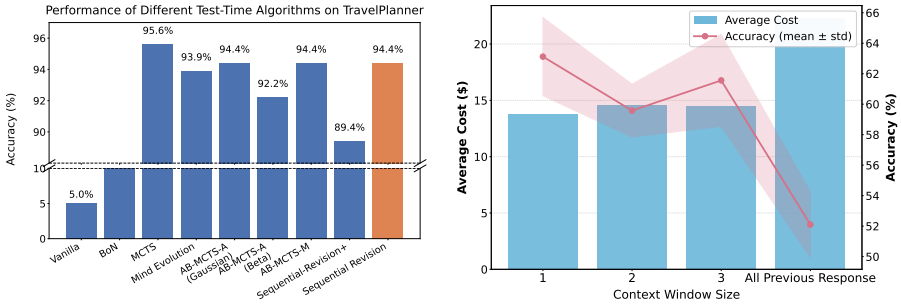



Figure 6: DeepSeek-V3 on TravelPlanner. **Left:** sequential revision vs. Mind Evolution / MCTS / AB-MCTS. **Right:** cost-accuracy as the revision context window covers the $k$ most recent responses, $k \in \{1, 2, 3\}$, or the full history.

## D Single-Shot Performance Does Not Determine Sequential-Revision Performance

Consider a problem with $N > 1$ candidate answers and exactly one correct answer, uniformly distributed across problems. Let one policy always output the same candidate at every turn, and let another sample uniformly and independently from the $N$ candidates at each turn. Both policies have pass@1 equal to $1/N$. Under sequential revision with budget $K$, however, the first policy still succeeds with probability $1/N$, whereas the second succeeds unless all $K$ samples are wrong, with probability

$$1 - \left(1 - \frac{1}{N}\right)^K.$$

For $K > 1$, this is strictly greater than $1/N$. Thus, equal single-shot performance does not imply equal sequential-revision performance, and optimizing $J_{\mathrm{OneShot}}$ does not in general optimize $J_{\phi_{\mathrm{SR}}}$.

## E 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 K), 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.

## F Proof of the Recovery Decomposition (1)

*Proof.* We prove (1) through the intermediate identity $J_{\phi_{\mathrm{SR}}}(\theta) = \sum_{t=1}^{K} \mathbb{E}[\mathbf{1}\{\tau \geq t\}\, V_\pi(z_t)]$, in three steps.

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

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

Summing over $t \leq K$,

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

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

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

*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

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

□

## G 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 13 and Figure 14, the training prompts rely on structured supervision rather than explicit error feedback.

**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.

## H 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 $\times$ NVIDIA H200 GPUs.

**RL Baseline.** Our RL baseline is implemented using the official `Skywork-OR1` codebase (He et al., 2025). 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. Concretely, RL and REVES both use GRPO (Shao et al., 2024) with batch 256 prompts, group size 16, PPO mini-batch 256, learning rate $10^{-6}$, response cap 8,192 tokens, sampling temperature 1.0, group-normalized scalar rewards (no critic), and the clipped PPO loss over valid response tokens. The credit-assignment argument of Section 4 concerns *cross-turn* credit only; within a response, token-level credit is standard RLVR.

**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: PPO/GAE with a critic, mini-batch 128, learning rate $10^{-6}$, two turns, 4 trajectories per prompt, and 512 prompts per update, so that each update draws 4,096 sampled responses as REVES does (compute accounting in Appendix I.1).

**SCoRe.** We use the SCoRe implementation in the same verl fork (full fine-tuning, no LoRA) with its two stages chained: Stage I (turn-2 reward with a turn-1 KL anchor) followed by Stage II (both-turn reward with correction shaping and KL), for a total of 320 gradient steps on the same Skywork-OR1 data and Qwen2.5-7B base, matching REVES's step budget.

**pass@$k$-trained policy.** We train Qwen2.5-7B with the pass@$k$ policy-optimization advantage of PKPO (Walder & Karkhanis, 2025) ($k = 8$, group size 16) using the Skywork hyperparameters above and the paper's graders, on a verl 0.4 stack (the Skywork fork does not run on the B200 GPUs used for these runs); the checkpoint at step 280 is compute-matched to REVES within a few percent.

**REVES (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 $K = 8$. The training prompts used for augmentation follow the templates described in Appendix G. 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 (`https://anonymous.4open.science/r/REVE-0D68`). 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. AIME24/25 numbers average 16 evaluation seeds and MATH500 8. Coding for the SCoRe, pass@$k$, and ablation rows is graded with the official LiveCodeBench evaluator (release v6, 272 problems from 2024-08 to 2025-01); for SR runs a single final program per problem is submitted to the grader.

# I METHOD DETAILS: MULTI-TURN COMPARISON, EFFICIENCY, AND TEST-TIME SENSITIVITY

## I.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, group-normalizes their final rewards $R_{1:G}$ into trajectory advantages $A_{1:G}$, and broadcasts $A_g$ uniformly to every turn in trajectory $g$. 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.

**Compute accounting.** Table 4 gives the accounting behind the Multi-Turn baseline of Tables 1 and 2: Multi-Turn and REVES draw the same 4,096 responses per update and generate similar total tokens; the Multi-Turn checkpoint is evaluated at 260 updates (13 more than REVES) and still generates slightly fewer tokens, so REVES's Oracle-32 margin of $+15.5$/$+15.2$/$+7.6$ (AIME24/AIME25/MATH500) is not a sampling-budget artifact.

Table 4: Compute accounting of the Multi-Turn baseline vs. REVES (Qwen2.5-7B, 8×H200).

| | **Multi-Turn** | **REVES** |
|---|---|---|
| Prompts × samples per update | $512 \times 4 \times 2$ turns | $256 \times 16$ |
| Sampled responses per update | 4,096 | 4,096 |
| Mean generated tokens per update | ≈3.6M | ≈3.9M |
| Optimizer updates (evaluated checkpoint) | 260 | 247 (3 refresh iterations) |
| Realized generated tokens | ≈0.94B | ≈0.97B |
| Wall-clock | 48h 0m | 30h 2m |

## I.2 REVISION AND VERIFICATION PROMPT ABLATION

Table 5 isolates the roles of revision and verification prompts in sequential revision training. RevisionOnly augments RL-generated data using only revision prompts, VerificationOnly uses only verification prompts, and REVES combines both. The Oracle-4 results show that revision prompts primarily improve *sequential revision capability*, whereas verification prompts train the model to assess solution validity.

To isolate the latter effect, 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 averaged over problems with sufficient correct samples. REVES achieves a higher AUROC (74.1%) than RevisionOnly (72.1%), showing that verification prompts improve confidence calibration. Combining the two prompt types therefore improves both revision capability and confidence estimation.

Table 5: Effect of revision and verification prompts on AIME25 (Qwen2.5-7B): 1-shot (OneShot), O-4 (Oracle-4), and SC-4 (SelfConf-4) accuracy for *RL* (no augmentation), *VerificationOnly*, *RevisionOnly*, and *REVES* (both).

| Model | 1-shot | O-4 | SC-4 |
|---|---|---|---|
| Qwen-2.5-7B-RL | **10.4** | 13.4 | 12.5 |
| Qwen-2.5-7B-VerificationOnly | 6.3 | 10.3 | 7.1 |
| Qwen-2.5-7B-RevisionOnly | 7.9 | 16.5 | 8.3 |
| Qwen-2.5-7B-REVES | 9.5 | **21.2** | **15.8** |

### I.3 Which states to train on: filter and feedback ablations

Table 6 varies only the Stage-I selection rule of Section 4.3, with three closed-loop iterations (augment, one epoch, re-augment) and everything else identical. *Unfiltered* keeps every visited state, including those of never-successful trajectories; it trails REVES on all four benchmarks while costing about $4\times$ the training time (its augmentation set is $\sim 12\times$ larger), so the successful-trajectory filter is a quality gain rather than a compute shortcut. *Difficulty-matched replay* keeps REVES's difficulty profile but drops the success condition and also trails, so the gain comes from the success condition itself, not from the difficulty mix. *Detailed feedback* replaces the binary training verdict with execution errors for coding; it does not improve over binary feedback, even in-domain on LiveCodeBench (differences within run-to-run noise, $\approx \pm 2.7$ at $n$=272), so the simpler binary training signal is retained. The three variants share one instrumented Stage-I generation; each variant's prompt set is built offline and trained with GRPO on $8\times$B200.

Table 6: Stage-I state-selection and feedback ablations (Qwen2.5-7B). Cells are revision / OneShot accuracy: Oracle-32 on math, TC-32 on LiveCodeBench (official grader). Train is total wall-clock on $8\times$B200. ‡: lower bound ($5/272$ programs ungradeable).

| Variant | AIME24 | AIME25 | MATH500 | LiveCodeBench | Train |
|---|---|---|---|---|---|
| Unfiltered (all visited states) | 41.9 / 15.8 | 39.2 / 11.0 | 93.1 / 74.6 | 26.1‡ / **20.9** | ≈109h |
| Difficulty-matched replay (no success cond.) | 40.0 / 13.5 | 38.5 / 7.9 | 93.5 / 75.0 | 29.0 / 17.0 | ≈28h |
| Detailed execution feedback (coding) | 42.5 / **15.2** | 40.4 / 7.5 | 93.8 / 74.8 | 27.2 / 17.7 | ≈27h |
| REVES (success-filtered, binary feedback) | **45.7** / 14.9 | **40.5** / 9.3 | **94.7 / 76.8** | **29.5** / 19.5 | ≈27h |

### I.4 SCoRe vs. REVES across revision budgets

Figure 9 sweeps the oracle budget $K \in \{2, 4, 8, 16, 32\}$ for SCoRe and REVES (Qwen2.5-7B, same data and step budget). Oracle-$K$ is the fraction of problems solved by round $\leq K$, derived from the same reflection records as Table 2. SCoRe's objective is defined for two turns and its curve flattens earlier; REVES keeps converting later revision rounds into successes.

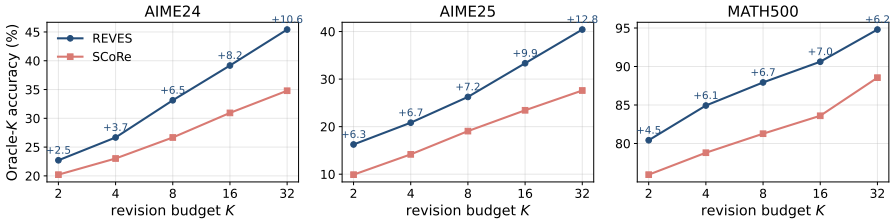



Figure 9: Oracle-$K$ accuracy of SCoRe and REVES (Qwen2.5-7B, same data and step budget) as the revision budget $K$ grows; annotations give REVES−SCoRe. The gap widens monotonically with $K$ on all three benchmarks.

### I.5 Continual augmentation and augmentation length

Figure 10 shows two consistent effects. Re-augmenting throughout training outperforms augmenting only once on both AIME24 and MATH500, because the training states continue to reflect the evolving policy's failures. Allowing longer Stage-I trajectories also improves accuracy, at additional generation cost.

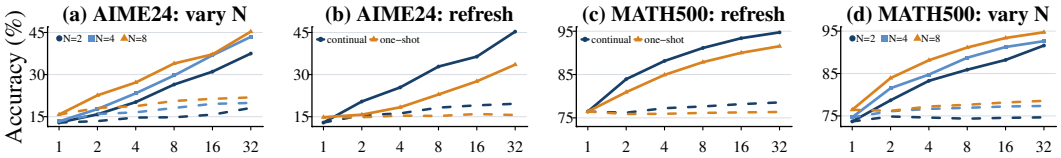



Figure 10: Accuracy versus test-time budget $K$: Oracle (solid) and SelfConf (dashed). **(a,d)** vary Stage-I augmentation length $N \in \{2, 4, 8\}$; **(b,c)** compare continual with one-shot augmentation.

### I.6 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 7). 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.

Table 7: 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 |

### I.7 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 8 reports accuracy and the average number 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.

## J 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.

Table 8: Sensitivity of test-time accuracy to the `TailConfidence` threshold $c$ on Qwen-2.5-7B-REVES 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 |

Table 9: 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 + REVES | 35.53 | **46.33** | **40.63** | **54.04** |

As shown in Table 9, our REVES 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.

## K Circle Packing Implementation Details

This section gives the implementation details behind the circle packing results in Figure 5(a). 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 (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.

**Reward shaping.** The raw objective of a candidate $y$ is its sum of radii $g(y) := \sum_{i=1}^{n} r_i(y)$ (a continuous quantity), which clusters in a narrow band near the optimum ($g \approx 2.63$ 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. Let $v(y) \geq 0$ denote the total constraint violation (circle overlap plus boundary excess), so that $y$ is feasible iff $v(y) = 0$. First, define the raw score $s(y)$ as

$$s(y) = \begin{cases} g(y), & v(y) = 0, \\ g(y) - v(y), & v(y) > 0 \text{ and } g(y) \leq 2.64, \\ -1, & v(y) > 0 \text{ and } g(y) > 2.64, \end{cases}$$

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 $\pi_\theta$ 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 11 visualizes one of the configurations REVES discovers, and Figure 12 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.

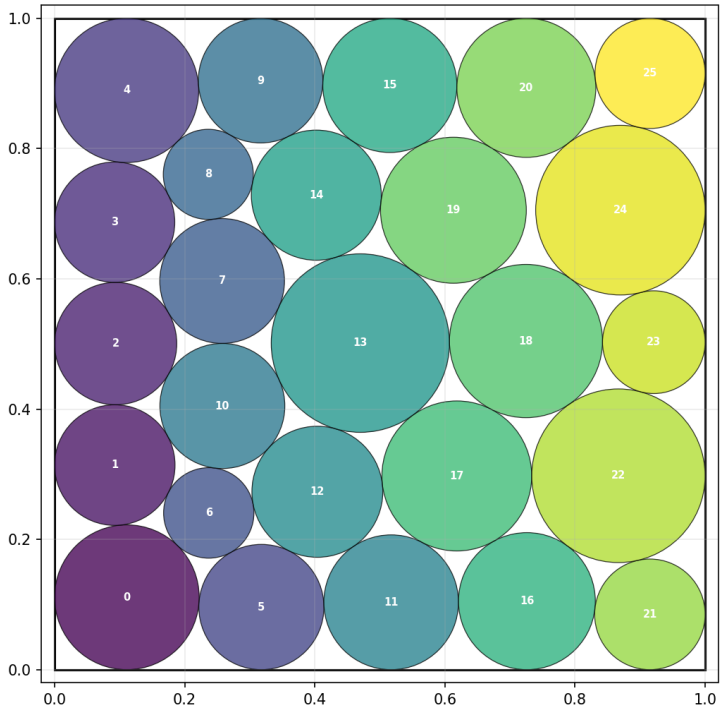



Figure 11: A circle packing configuration found by REVES on the $n = 26$ instance, achieving sum-of-radii $= 2.635983$.

**Circle-Packing Program**

```python
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):
```

```python
        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]

    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]
        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))
```

Figure 12: Python program output by the REVES-trained Qwen3-4B policy that produces the configuration in Figure 11.

**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 ...

```
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 13: 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 14: Prompt template for the code verification stage. The verifier evaluates a candidate solution and outputs a binary correctness verdict.

## L Extended Related Work

This section expands the positioning of Section 2 with per-method comparisons.

**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 & 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. The closest methods are SCoRe (Kumar et al., 2024) and ExIt (Jiang et al., 2026a). SCoRe jointly optimizes a two-turn trajectory with a turn-1 KL anchor and correction shaping, so training is defined for depth one; REVES trains on failed states at every depth of a budget-$K$ trajectory with single-turn RL, and its advantage over SCoRe grows with the revision budget (Section 5). ExIt shares the single-step principle, training only on one-step self-improvement tasks and iterating at inference; REVES is not a curriculum variant of it but differs in what the single-step task is. ExIt's task is to improve a given solution (correct or not), rewarded by the normalized improvement $\max(0, \Delta/\Delta_{\max})$, with tasks harvested into a persistent buffer and prioritized by reward variance; REVES's task is to recover from a state that is known to be wrong, rewarded by verified correctness, with states drawn from the first-success objective it actually optimizes at test time (Eq. (1)), retained only from trajectories that recover within budget (Appendix I.3 shows this filter, not the state mix, drives the gain), refreshed every epoch with no buffer, and paired with verification prompts so the policy can stop without a verifier. Section 4.2 adds what neither method offers: a characterization of when this per-state estimator beats trajectory-level credit.

**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.

**Meta-RL views of repeated attempts.** LaMer (Jiang et al., 2026b) also frames repeated attempts as meta-RL, training an LLM agent on trials of $N$ sequential episodes with in-context self-reflection and a cross-episode discounted return optimized by a multi-turn policy gradient, so early episodes receive credit for later successes and rollouts are generated sequentially. REVES shares the framing but targets the first-success objective of a verifier-based controller through its exact per-state decomposition and trains decoupled single-turn recovery problems, so no cross-episode credit is broadcast; Section 4.2 quantifies exactly this trade between the two estimators.

## M REVES Pseudocode and Overview

Algorithm 2 gives one epoch of REVES exactly as implemented, including which responses become revision versus verification prompts. Figure 4 in the main text shows the loop.

**Algorithm 2** REVES (one epoch)

**Input:** policy $\pi_\theta$, RL prompt set $\mathcal{D}_0$, budget $K$, verifier $r^\star$
**Stage I (augmentation).** $\mathcal{D}_{\text{rev}} \leftarrow \emptyset$, $\mathcal{D}_{\text{ver}} \leftarrow \emptyset$
**for** each $x \in \mathcal{D}_0$ **do**
  run SR under $\pi_\theta$ with budget $K$; obtain $(y_1, \ldots, y_\ell)$ with $\ell = \tau$
  **if** $r^\star(x, y_1) = 0$ and $r^\star(x, y_\ell) = 1$ *// keep only trajectories wrong at $y_1$ that recover within $K$* **then**
    $\mathcal{D}_{\text{rev}} \leftarrow \mathcal{D}_{\text{rev}} \cup \{\texttt{RevisionPrompt}(x, y_i)\}_{i=1}^{\ell-1}$ *// incorrect attempts only*
    $\mathcal{D}_{\text{ver}} \leftarrow \mathcal{D}_{\text{ver}} \cup \{(\texttt{VerificationPrompt}(x, y_i),\ r^\star(x, y_i))\}_{i=1}^{\ell}$ *// $y_\ell$ labeled True*
  **end if**
**end for**
**Stage II (single-turn RL).** Run GRPO for one epoch on $\mathcal{D}_0 \cup \mathcal{D}_{\text{rev}} \cup \mathcal{D}_{\text{ver}}$: for each prompt draw a group of fresh responses from $\pi_\theta$, reward with $r^\star$ (revision) or verification accuracy (verification), and update $\theta$.
**Output:** updated $\pi_\theta$ (used as the rollout policy of the next epoch)

## N Formal Statement of Theorem 4.1

**Definition N.1** (Surrogate revision chain)**.** There are $N$ transient states $z_1, \dots, z_N$ and one absorbing success state ✓ (reward 1); the only way to earn reward 0 is to exhaust the budget $K$. Row $i$ of the transition matrix is a softmax over the actions $\{\text{succeed}\} \cup \{\text{move-to-}j\}_{j=1}^{N}$ with logits $\theta_i \in \mathbb{R}^{N+1}$, so one action draw sets both the recovery probability $P_i := \pi_i(\text{succeed}) = V_\pi(z_i)$ and the transition row $Q_{ij} := \pi_i(\text{move-to-}j)$. Trajectories start at $z_1$; the score of action $a$ at state $i$ is $g_i(a) = \nabla_{\theta_i} \log \pi_i(a) = e_a - \pi_i$. With $S_K = \sum_{t=0}^{K-1} Q^t$ and $\mu = e_1$, the visit counts and objective are $\rho = \mu^\top S_K$ and $J = \sum_i \rho_i P_i = \Pr(\text{success within } K)$, and $\nabla J = \sum_i \rho_i \nabla P_i + \sum_i P_i \nabla \rho_i =: \text{(I)} + \text{(II)}$.

**The two estimators.** To compare both estimators against the exact $\nabla J$, we analyze their unnormalized REINFORCE forms. Multi-turn RL rolls out $M$ trajectories and broadcasts each terminal reward $R(\zeta) \in \{0, 1\}$ to every visited score; REVES rolls out one trajectory to obtain visitation indicators $B_i = \mathbf{1}\{z_i \text{ visited}\}$ and draws $M$ fresh actions at each visited state with the immediate reward $r^\star(a) = \mathbf{1}\{a = ✓\}$:

$$\hat{g}^{\mathrm{MT}} = \frac{1}{M} \sum_{m=1}^{M} R(\zeta^{(m)}) \sum_{t=1}^{\tau_m} g_{s_t^{(m)}}\big(a_t^{(m)}\big), \qquad \hat{g}_i^{\mathrm{RE}} = B_i \cdot \frac{1}{M} \sum_{l=1}^{M} r^\star\big(a^{(l)}\big)\, g_i\big(a^{(l)}\big). \tag{3}$$

The practical implementations replace these raw rewards with group-normalized GRPO advantages: MultiTurn shares one trajectory advantage across all turns, whereas REVES computes per-response advantages separately within the fresh group at each state. The theorem below does not model this normalization; it isolates the same cross-turn versus per-state credit structure in estimators whose expectations can be compared exactly with $\nabla J$.

**Theorem N.2** (Bias–variance crossover, formal)**.** *Let $w_i := \Pr(B_i = 1)$, $a_{\mathrm{MT}} := \operatorname{Tr}\operatorname{Cov}(\hat{g}^{\mathrm{MT}}_{M=1})$ be the trace covariance of one trajectory-REINFORCE sample, $c_{\mathrm{RE}} := \sum_i w_i P_i (1 - P_i) \|e_✓ - \pi_i\|^2$,*

$$\beta := \sum_i (w_i - \rho_i) \nabla_{\theta_i} P_i - \sum_i P_i \nabla \rho_i, \qquad F := \|\beta\|^2 + \sum_i w_i (1 - w_i) \|\nabla_{\theta_i} P_i\|^2.$$

*Then $\mathbb{E}[\hat{g}^{\mathrm{MT}}] = \nabla J$ and $\mathbb{E}[\hat{g}_i^{\mathrm{RE}}] = w_i \nabla_{\theta_i} P_i$, so REVES estimates term* (I) *only, with bias $\beta$ (equal to $-$(II) when no state is revisited), and for the chain of Definition N.1*

$$\mathrm{MSE}(\hat{g}^{\mathrm{MT}}) = \frac{a_{\mathrm{MT}}}{M}, \qquad \mathrm{MSE}(\hat{g}^{\mathrm{RE}}) = F + \frac{c_{\mathrm{RE}}}{M}. \tag{4}$$

*If $F > 0$ and $a_{\mathrm{MT}} > c_{\mathrm{RE}}$, the two curves cross exactly once, at $M^\star = (a_{\mathrm{MT}} - c_{\mathrm{RE}})/F$: REVES has lower MSE for $M < M^\star$ and multi-turn RL for $M > M^\star$. If $a_{\mathrm{MT}} \le c_{\mathrm{RE}}$, multi-turn RL has lower MSE for every $M$.*

## O Proof of Theorem N.2

*Proof of Theorem N.2. Means.* (a) The trajectory distribution is a product of policy probabilities $\prod_t \pi_{s_t}(a_t)$ (stopping is a deterministic function of the actions), and $J = \mathbb{E}[R(\zeta)]$; the score-function identity gives $\nabla J = \mathbb{E}[R(\zeta) \sum_t \nabla \log \pi_{s_t}(a_t)] = \mathbb{E}[\hat{g}^{\mathrm{MT}}_{M=1}]$, and averaging $M$ i.i.d. trajectories preserves the mean. (b) Conditional on $B_i = 1$, the $M$ actions are i.i.d. from $\pi_i$, and $\mathbb{E}_{a \sim \pi_i}[r^\star(a) g_i(a)] = \sum_a \pi_i(a) r^\star(a) \nabla_{\theta_i} \log \pi_i(a) = \nabla_{\theta_i} \sum_a \pi_i(a) r^\star(a) = \nabla_{\theta_i} P_i$; conditional on $B_i = 0$ the block is zero. Hence $\mathbb{E}[\hat{g}_i^{\mathrm{RE}}] = w_i \nabla_{\theta_i} P_i$. Since $\nabla J = \sum_i \rho_i \nabla P_i + \sum_i P_i \nabla \rho_i$ and $\nabla P_i$ is supported on block $i$, $\beta = \sum_i (w_i - \rho_i) \nabla P_i - \sum_i P_i \nabla \rho_i$; if no state is revisited then $w_i = \rho_i$. Note also that $r^\star(a) g_i(a) = \mathbf{1}\{a = ✓\}(e_✓ - \pi_i)$ is a Bernoulli$(P_i)$ multiple of a fixed vector, with trace covariance $P_i(1 - P_i)\|e_✓ - \pi_i\|^2$.
*MSE.* $\mathrm{MSE}(\hat{g}) = \|\mathbb{E}\hat{g} - \nabla J\|^2 + \operatorname{Tr}\operatorname{Cov}(\hat{g})$. For $\hat{g}^{\mathrm{MT}}$ the bias is zero by (a), and the trace covariance of an average of $M$ i.i.d. samples is $a_{\mathrm{MT}}/M$. For $\hat{g}^{\mathrm{RE}}$ the bias is $\beta$. The trace covariance is the sum over coordinates of variances, hence the sum over blocks $\sum_i \operatorname{Tr}\operatorname{Cov}(\hat{g}_i^{\mathrm{RE}})$ (cross-block covariances do not enter the trace). By the law of total covariance conditioning on $B_i$,

$$\begin{aligned} \operatorname{Tr}\operatorname{Cov}(\hat{g}_i^{\mathrm{RE}}) &= \mathbb{E}\big[\operatorname{Tr}\operatorname{Cov}(\hat{g}_i^{\mathrm{RE}} \mid B_i)\big] + \operatorname{Tr}\operatorname{Cov}\big(\mathbb{E}[\hat{g}_i^{\mathrm{RE}} \mid B_i]\big) \\ &= w_i \frac{P_i(1 - P_i)\|e_✓ - \pi_i\|^2}{M} + w_i(1 - w_i)\|\nabla_{\theta_i} P_i\|^2, \end{aligned}$$

using the trace covariance above for the first term and $\mathbb{E}[\hat{g}_i^{\mathrm{RE}} \mid B_i] = B_i \nabla_{\theta_i} P_i$ for the second. Summing over $i$ gives $\mathrm{MSE}(\hat{g}^{\mathrm{RE}}) = F + c_{\mathrm{RE}}/M$. Then $\mathrm{MSE}(\hat{g}^{\mathrm{MT}}) < \mathrm{MSE}(\hat{g}^{\mathrm{RE}})$ iff $(a_{\mathrm{MT}} - c_{\mathrm{RE}})/M < F$; if $a_{\mathrm{MT}} \le c_{\mathrm{RE}}$ this holds for every $M$ (as $F > 0$), and otherwise iff $M > M^\star := (a_{\mathrm{MT}} - c_{\mathrm{RE}})/F$. □

## P Numerical Details of the Surrogate

**Exact gradient.** With $S_K = \sum_{t=0}^{K-1} Q^t$ and $\mu = e_1$, $\rho = \mu^\top S_K$ and $J = \rho^\top P$ are exact at finite $K$, and differentiating the recursion $S_{t+1} = I + QS_t$ gives $\nabla\rho$ and hence both terms of $\nabla J$ in closed form. All identities were checked against central finite differences (agreement to $10^{-9}$), and the closed-form MSE of Theorem N.2 against a direct Monte-Carlo estimate of $\mathbb{E}\|\hat{g}_M - \nabla J\|^2$ ($2 \times 10^6$ single-sample draws for the moments, 4,000 replicates per $M$), which agree to two to three significant figures; these Monte-Carlo estimates are the markers in Figure 3(a).

Table 10: MSE at $K$=8: closed form from (4) versus a direct Monte-Carlo estimate of $\mathbb{E}\|\hat{g}_M - \nabla J\|^2$, with ground-truth $\nabla J$ computed by finite differences. Multi-turn RL's MSE is pure variance and vanishes with $M$, whereas REVES approaches the bias floor $F = 0.043$. The crossover occurs near $M$=30.

| | $\mathrm{MSE}(\hat{g}^{\mathrm{MT}})$ | | $\mathrm{MSE}(\hat{g}^{\mathrm{RE}})$ | | |
|---|---|---|---|---|---|
| $M$ | closed form | direct MC | closed form | direct MC | winner |
| 1 | 1.697 | 1.695 | 0.472 | 0.482 | REVES |
| 2 | 0.849 | 0.843 | 0.257 | 0.259 | REVES |
| 4 | 0.424 | 0.429 | 0.150 | 0.148 | REVES |
| 8 | 0.212 | 0.211 | 0.097 | 0.096 | REVES |
| 16 | 0.106 | 0.106 | 0.070 | 0.070 | REVES |
| 32 | 0.053 | 0.053 | 0.056 | 0.056 | Multi-turn |
| 64 | 0.027 | 0.027 | 0.050 | 0.050 | Multi-turn |

Table 11: Horizon dependence for the reference chain. As $K$ grows, the crossover $M^\star$ moves to a larger sampling budget before saturating, while REVES's MSE floor and relative bias also increase.

| $K$ | $J$ | $a_{\mathrm{MT}}$ | REVES floor | $\|\beta\|/\|\nabla J\|$ | $M^\star$ |
|---|---|---|---|---|---|
| 2 | 0.306 | 0.38 | 0.017 | 0.13 | 6.8 |
| 4 | 0.554 | 0.88 | 0.031 | 0.36 | 15.0 |
| 8 | 0.815 | 1.70 | 0.043 | 0.69 | 29.7 |
| 12 | 0.924 | 2.33 | 0.055 | 1.35 | 34.7 |
| 20 | 0.987 | 2.99 | 0.080 | 6.10 | 32.1 |

**Instances behind Figure 3.** The reference chain of panel (a) has $N = 4$ states with recovery probabilities $P = (0.165, 0.164, 0.165, 0.221)$, success several moves downstream of $z_1$, and $K = 8$ ($J = 0.815$), giving $a_{\mathrm{MT}} = 1.70$, $c_{\mathrm{RE}} = 0.43$, $F = 0.043$ and $M^\star \approx 30$. Panels (b) and (c) use homogeneous chains with $P_i \equiv p$ and uniform $Q$ ($N = 4$); (b) fixes $K = 8$ and varies $p$, (c) varies $K$ for several $p$. Because moves can go backward and $K > N$, states are revisited, so REVES's single-trajectory indicator estimates the visit probability $w_i$ rather than the count $\rho_i$; this is the $(w_i - \rho_i)\nabla P_i$ term in the bias $\beta$ of Theorem N.2. The scripts that generate every number and figure of Section 4.2 are in the released repository.