CAN LANGUAGE MODELS LEARN FROM FAILURES ALONE AT TEST TIME?

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time training (TTT), where language models (LMs) undergo parameter updates during inference, has emerged as a promising approach for adapting LMs to individual problems. Prior work on problem-specific TTT often operates in hill-climbing settings, where candidate solutions can be evaluated with a fine-grained scalar reward. This paper asks: how far can test-time training go when learning only from failures, without such fine-grained feedback? We adapt a range of existing optimization methods to this regime and propose *Controlled Redistribution (*CoRed*)*, which treats failure-only TTT as controlled reshaping of the model's token-level predictive distribution. Across countdown games, math, and code tasks, CoRed improves both solvability and sample efficiency. On challenging math problems, CoRed enables Qwen3-4B-Instruct to solve more than $2\times$ as many problems as the frozen base model within 4,096 attempts, while substantially reducing the number of attempts required to reach the first correct solution on problems that the base model can eventually solve within 4,096 attempts. Our results demonstrate that substantial test-time adaptation is possible even when learning from failures alone.

1 INTRODUCTION

Modern inference paradigms increasingly trade additional computation for deeper problem solving, allowing models to search, refine, and adapt their reasoning on challenging individual instances (Wang et al., 2023b; Yao et al., 2023; Snell et al., 2025). Test-time training (TTT) takes this idea further by using inference-time compute not only to generate more samples, but also to update the model: the model repeatedly generates candidate solutions, evaluates them, and performs parameter updates on the same problem instance (Yuksekgonul et al., 2026; Akyürek et al., 2025).

Existing TTT applications typically operate in settings with fine-grained scalar feedback, such as continuous objectives measuring solution quality (e.g., runtime in GPU kernel engineering (Yuksekgonul et al., 2026)). As illustrated in Figure 1 (left), such feedback turns adaptation into a hill-climbing problem: even when all candidates fail, relative scores indicate which solutions are better. **We instead study a substantially different regime** as in Figure 1 (right), in which the model receives only failed trajectories, with no score or ranking among them. Before the first success, every candidate is simply rejected, providing no direct evidence about which behaviors should be reinforced or which failures are closer to success.

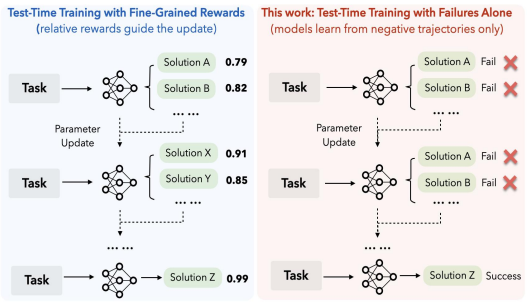

Figure 1: Illustration of the test-time training setting studied in this paper, where the model is iteratively updated using only failed trajectories, without access to a fine-grained scalar reward.

We take a first step toward studying whether TTT can still provide meaningful gains in this failure-only regime. We adapt a range of existing techniques, such as Unlikelihood Training (Welleck et al., 2019) and Self-Distillation Policy Optimization (Hübotter et al., 2026), and evaluate how effectively they can exploit all-negative trajectories. We further propose **Controlled Redistribution (CoRed)**, a framework designed specifically for failure-only TTT that views test-time adaptation as controlled

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.

reshaping of the model's token-level predictive distribution. CoRED decreases the probability of token choices appearing in failed trajectories, balances this negative pressure across sampled tokens with a reshaped unlikelihood learning objective, and uses a fixed base-policy KL constraint to control how probability mass is redistributed. Together, these components enable targeted exploration away from failed trajectories without drifting arbitrarily from the base policy.

We evaluate and consider failure-only TTT successful by asking whether it can 1) solve problems that the frozen base model rarely solves even after extensive sampling, and 2) reduce the number of attempts needed to find a first successful solution. These gains are particularly useful when verification is costly. Candidate solutions may require expensive execution-based evaluation, while more open-ended problems may depend on expert judgment for reliable verification. In such settings, reducing the number of failed attempts reduces the amount of verification required. Figure 2 illustrates this behavior: TTT solves more problems than repeated sampling from the frozen base model under the same attempt budget.

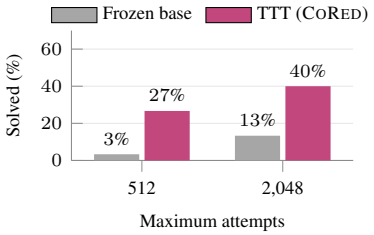

Figure 2: Solve rates of Qwen3-4B-Instruct on hard math problems at matched attempt budgets.

We evaluate on hard countdown, math, and coding tasks against frozen-model sampling and baselines adapted from unlikelihood training, negative sample reinforcement, entropy minimization, and self-distillation policy optimization (Welleck et al., 2019; Zhu et al., 2025; Gao et al., 2025; Hübotter et al., 2026). CoRED improves both solvability and sample efficiency across settings. On hard math problems, Qwen3-4B-Instruct (Yang et al., 2025) reaches a 46.7% solve rate within 4,096 attempts, compared with 20.0% for naive sampling. On hard Codeforces problems, the frozen model solves only 6.7% within 4,096 attempts, while CoRED solves 16.7% using around 800 attempts per problem. Analysis on Countdown further shows that failure-only adaptation meaningfully reshapes subsequent exploration. Overall, our results highlight the promise of TTT even without dense, scalar rewards.

Our contributions are: 1) We take the first step toward studying failure-only TTT and adapt existing methods to this setting. 2) We propose CoRED, which performs controlled redistribution of probability mass from failed rollouts. 3) We empirically show that CoRED improves both solve rate and sample efficiency on hard problems under failure-only TTT.

2 Problem Setup: Test-Time Training from Failures Alone

2.1 Preliminaries: Test-Time Training

We first set up the preliminaries for general test-time training (TTT). We focus on *instance-specific TTT*, which adapts a pretrained reference policy π_0 to a single test problem x. TTT initializes temporary adaptation parameters ϕ_0 such that $\pi_{\phi_0} = \pi_0$. At iteration s, the autoregressive policy generates a batch of candidate solutions $\mathcal{B}_s = \{y_{s,k}\}_{k=1}^{K}$, where $y_{s,k} \sim \pi_{\phi_s}(\cdot \mid x)$.

A problem-specific evaluator F_x provides the feedback dataset at iteration s:

$$\mathcal{D}_s = \{(y_{s,k}, F_x(y_{s,k}))\}_{k=1}^{K}.$$

Based on the observed feedback, the policy is updated according to a generic update-rule $\mathcal{U}$:

$$\phi_{s+1} = \mathcal{U}\left(\phi_s; \mathcal{D}_s, \pi_0\right), \tag{1}$$

Prominent Applications of TTT with a Fine-Grained Scalar Reward. Recent applications of TTT include mathematical discovery, kernel engineering, and algorithm optimization, where fine-grained scalar rewards distinguish candidate quality (Yuksekgonul et al., 2026; Georgiev et al., 2025; Zhang et al., 2025; Haugland, 2016).

Given the feedback dataset $\mathcal{D}_s$, a batch-centered policy-gradient update is

$$\mathcal{U}(\phi_s; \mathcal{D}_s, \pi_0) = \phi_s + \frac{\eta}{K} \sum_{k=1}^{K} \big(F_x(y_{s,k}) - \bar{F}_s\big) \nabla_{\phi_s} \log \pi_{\phi_s}(y_{s,k} \mid x), \tag{2}$$

where $\bar{F}_s = K^{-1} \sum_{k=1}^{K} F_x(y_{s,k})$ and $\eta > 0$ is the learning rate. When all batch rewards are identical, the centered gradient estimate is zero.

Hence, a fine-grained scalar reward provides a contrastive optimization signal among candidates, and existing methods rely on variation in $F_x(y)$ to distinguish better from worse candidates.

2.2 Test-Time Training from Failures Alone

We then specialize the TTT framework to a stricter regime, failure-alone TTT, where feedback is restricted to a binary verifier $F_x(y) \in \{0, 1\}$ indicating rejection or acceptance, respectively. The procedure terminates upon finding an accepted candidate and returns a verified solution. Thus, every update uses an all-negative batch:

$$\mathcal{B}_s^- = \{y_{s,k}\}_{k=1}^{K}, \qquad F_x(y) = 0 \quad \text{for all } y \in \mathcal{B}_s^-. \tag{3}$$

These batches before the first success provide no reward-based ranking among candidates: $F_x(y_{s,k}) = \bar{F}_s = 0$ for every k. Consequently, the centered gradient estimate in Eq. 2 vanishes, leaving the policy unchanged. The challenge is therefore to extract useful adaptation signals from rejection alone, without successful trajectories or fine-grained rewards.

Evaluation protocol. Our objective is to reduce the number of attempts required to reach the first success. We order all evaluated rollouts as $y_1, y_2, \ldots$ and define the first-success time

$$T^\star(x) = \inf \{n \geq 1 : F_x(y_n) = 1\}, \tag{4}$$

with $T^\star(x) = \infty$ if no verified solution is found. In other words, we aim at improving sample efficiency by reducing $T^\star(x)$, represented as required sample attempts before the first success.

2.3 Existing Methods under Failure-Only Feedback

While TTT from failure only has been under-explored in the literature, we instantiate several existing technique under the failure-only setting. These adapted techniques differ in how they convert rejected rollouts into a parameter update. We show the objective functions $\mathcal{L}$ that allow them to learn from negative rollouts. For the following objectives, we write $y_i = y_{s,i}$ and define $p_{i,t}^{\phi}(v) = \pi_\phi(v \mid x, y_{i,<t})$ and $p_{i,t}^{\phi} = p_{i,t}^{\phi}(y_{i,t})$; $\langle \cdot \rangle_{\mathcal{B}_s^-}$ denotes the uniform average over all token positions in $\mathcal{B}_s^-$.

Naive Sampling. The simplest approach performs no adaptation and repeatedly samples from the frozen reference policy π_0. I.e., $y_n \sim \pi_0(\cdot \mid x),\ n = 1, \ldots, N$.

Unlikelihood Training (UL) (Welleck et al., 2019) applies token-level unlikelihood loss to a rejected rollout:

$$\mathcal{L}_{\mathrm{UL}}\left(\phi; \mathcal{B}_s^-\right) = \left\langle -\log\left(1 - p_{i,t}^{\phi}\right) \right\rangle_{\mathcal{B}_s^-}.$$

Minimizing this objective decreases the probability of each token sampled along a failed trajectory.

Negative Sample Reinforcement (NSR) (Zhu et al., 2025) assigns reward -1 to each rejected rollout. Following its token-level gradient analysis, the objective is given as:

$$\mathcal{L}_{\mathrm{NSR}}\left(\phi; \mathcal{B}_s^-\right) = \left\langle p_{i,t}^{\phi} \right\rangle_{\mathcal{B}_s^-}.$$

Entropy Minimization (EM) (Gao et al., 2025) sharpens the model's predictive distribution at prefixes encountered in the sampled rollouts. Define $H_{i,t}^{\phi} = -\sum_{v \in \mathcal{V}} p_{i,t}^{\phi}(v) \log p_{i,t}^{\phi}(v)$, where $\mathcal{V}$ is the vocabulary. We adapt the objective to the failure-only protocol by minimizing entropy on prefixes from the rejected batch:

$$\mathcal{L}_{\mathrm{EM}}\left(\phi; \mathcal{B}_s^-\right) = \left\langle H_{i,t}^{\phi} \right\rangle_{\mathcal{B}_s^-}.$$

Self-distillation Policy Optimization (SDPO) (Hübotter et al., 2026) trains an unconditioned policy to match a stop-gradient teacher policy conditioned on feedback. The original method assumes that feedback can be represented as text and may contain detailed environment information. To instantiate SDPO under failure-only feedback, let f_i^- denote a textual hint of the binary verifier result

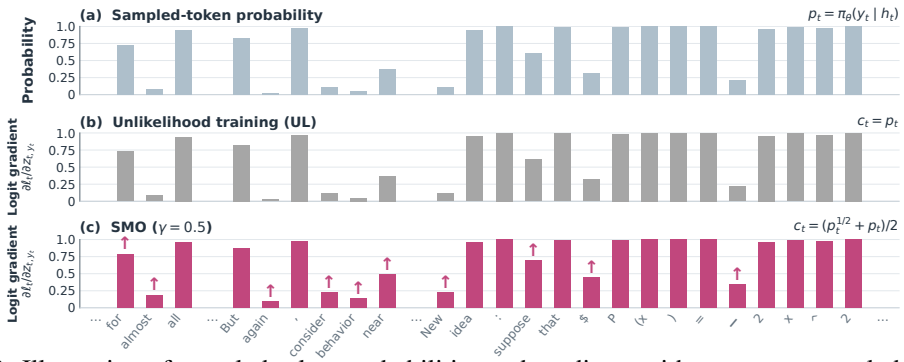

Figure 3: Illustration of sampled-token probabilities and gradients with respect to sampled-token logits in a rejected reasoning trajectory. UL predominantly penalizes high-confidence tokens, even when they represent locally correct computations, while placing little negative pressure on lower-probability choices. SMO better balances this pressure by upweighting lower-probability tokens.

associated with y_i (i.e., a feedback that a solution is incorrect). Define the feedback-conditioned teacher distribution

$$q_{i,t}^{\phi}(v) = \pi_{\phi}\left(v \mid x, f_i^{-}, y_{i,<t}\right).$$

The SDPO objective minimizes the KL between the feedback-conditioned teacher and student.

$$\mathcal{L}_{\text{SDPO}}\left(\phi; \mathcal{B}_s^{-}\right) = \left\langle \text{KL}\left(p_{i,t}^{\phi}(\cdot) \,\middle\|\, \text{stopgrad}\left[q_{i,t}^{\phi}(\cdot)\right]\right)\right\rangle_{\mathcal{B}_s^{-}}.$$

3 Method: Controlled Redistribution

With only failure feedback, a natural strategy is to *unlearn* rejected trajectories by decreasing their probability. We propose Controlled Redistribution (CoRed), which follows this principle while adapting unlikelihood training (UL) to better suit failure-only TTT. At a high level, we consider two factors in effectively redistributing probability mass under failure-only supervision: (1) how strongly sampled tokens should be suppressed as a function of their current probability, and (2) how the removed probability mass should be redistributed in the absence of any positive supervision. We first examine a limitation of vanilla UL in this setting, which motivates our two proposed components: a smoothed unlikelihood objective (§ 3.1) and a fixed base-policy KL constraint (§ 3.2).

Limitations of Unlikelihood Training. We begin by analyzing why updates of vanilla UL can be undesirable for failure-only TTT, particularly on reasoning-intensive tasks. The key issue is that **UL applies stronger negative pressure to sampled tokens that the model currently assigns higher probability**. In a rejected reasoning trajectory, however, many such high-confidence tokens may correspond to locally correct computations, common syntax, or otherwise reliable steps rather than the source of the final failure.

Specifically, let $p_j = \text{softmax}(z)_j$ denote the probability of token j at a sampled context. For a sampled token y, the unlikelihood loss $\ell_{\text{UL}}(p_y) = -\log(1 - p_y)$ has sampled-token logit gradient

$$\frac{\partial \ell_{\text{UL}}}{\partial z_y} = p_y. \tag{5}$$

Thus, the sampled-token logit is suppressed in proportion to the token's probability.

This behavior is potentially problematic when supervision is available only at the trajectory level. Reasoning traces often contain many highly predictable tokens that are locally correct despite appearing in an ultimately incorrect solution. In math problems, these may include routine equations and calculations. For example, Figure 3 shows intermediate mathematical tokens such as $P(x) = -2x^2$ whose probabilities approach 1. Similarly, in coding problems, syntax and repeated structural patterns can be assigned very high probability. Vanilla UL therefore concentrates a substantial fraction of its negative pressure on these confident tokens, even though they may not be responsible for the failure. Meanwhile, lower-probability tokens—which often occur at more uncertain decision points where alternative reasoning branches are possible (Wang et al., 2025c)—receive weak updates.

3.1 Reshaping Unlikelihood Objective

The limitations of vanilla UL motivate modifying how negative pressure depends on the probability of a sampled token. Our first control therefore reshapes this dependence to place relatively greater pressure on low-probability sampled tokens. To achieve this, we introduce the **smoothed unlikelihood objective** (SMO):

$$\ell_{\mathrm{SMO}}(p_y;\gamma) = -\log(1-p_y^{\gamma}), \qquad 0<\gamma\leq 1, \tag{6}$$

which recovers vanilla UL when $\gamma = 1$. For the sampled-token logit z_y, the gradient is

$$\frac{\partial \ell_{\mathrm{SMO}}}{\partial z_y} = c_\gamma(p_y) = \frac{\gamma p_y^{\gamma}(1-p_y)}{1-p_y^{\gamma}}. \tag{7}$$

For $0<\gamma<1$, $c_\gamma(p_y)$ is larger than the corresponding UL gradient p_y for $p_y \in (0,1)$, while approaching the same limit as $p_y \to 1$. Importantly, the amplification is concentrated on low-probability sampled tokens and diminishes as p_y increases. As illustrated in Figure 3, SMO substantially upweights lower-probability sampled choices. Thus, SMO redistributes update strength toward less confident decisions rather than concentrating penalties on confident, locally correct tokens.

For a concrete example, when $\gamma = \frac{1}{2}$, the sampled-token logit gradient and its ratio to the UL gradient simplify to

$$c_{1/2}(p_y) = \frac{p_y^{1/2}+p_y}{2}, \qquad \frac{c_{1/2}(p_y)}{p_y} = \frac{1+p_y^{-1/2}}{2}.$$

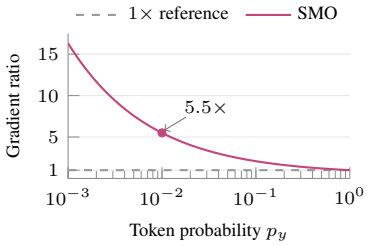

Figure 4: SMO-to-UL gradient ratio for $\gamma = 0.5$.

Figure 4 visualizes this ratio as a function of token probability on a logarithmic axis. At $p_y = 10^{-2}$, SMO produces a gradient $5.5\times$ that of UL, whereas at $p_y = 0.99$, the ratio is only approximately 1.003. Thus, SMO selectively amplifies updates on low-probability sampled choices while preserving nearly the same pressure as UL on highly confident tokens. This yields a more balanced distribution of negative pressure across tokens with different confidence levels. **We set γ to be 0.5 throughout our experiments.**

Let $\mathcal{B}_s^-$ denote the batch of rejected trajectories at test-time step s, N_i the number of generated tokens in trajectory i, and $h_{i,t} = (x, y_{i,<t})$ the context at position t. We apply SMO to every sampled token in each rejected trajectory, resulting in the batch-level objective

$$\mathcal{L}_{\mathrm{SMO}} = \frac{1}{|\mathcal{B}_s^-|}\sum_{i=1}^{|\mathcal{B}_s^-|}\frac{1}{N_i}\sum_{t=1}^{N_i}\ell_{\mathrm{SMO}}(\pi_{\phi_s}(y_{i,t}\mid h_{i,t});\gamma)\,. \tag{8}$$

Alternative Reshaping via Entropy Gating. While SMO provides a smooth and general mechanism for reshaping negative pressure across token probabilities, controlled redistribution can also be implemented more aggressively by explicitly preventing updates to highly confident tokens.

Recent work suggests that high-entropy tokens often mark branching or planning decisions, while low-entropy tokens tend to reflect deterministic execution steps (Wang et al., 2025c;b). This motivates an alternative **entropy-gated** variant. We apply the negative objective only at the 20% of sampled positions with the highest entropy and mask its gradients elsewhere. This avoids directly penalizing masked tokens in routine arithmetic or other locally deterministic computations.

We find SMO to work well generally across settings, while entropy gating can be particularly effective when the token distribution is highly skewed and trajectories contain a large fraction of near-deterministic, high-confidence tokens. See § 4.4 for further analysis.

3.2 Base-Policy KL Constraint

Failure-only feedback provides no guidance on which alternatives are preferable, so repeated on-policy updates can cause the adapted policy to drift toward poorly supported regions. To limit this drift, we use a KL constraint to keep the adapted policy close to the fixed base policy π_0 at the sampled contexts. For each position, let $p_{i,t}^{\phi_s}(v) = \pi_{\phi_s}(v \mid h_{i,t})$ and $p_{i,t}^{0}(v) = \pi_{\phi_0}(v \mid h_{i,t})$ denote the adapted and base-policy distributions, respectively. We define the base-policy KL constraint

$$\mathcal{L}_{\mathrm{KL}} = \left\langle D_{\mathrm{KL}}\left(p_{i,t}^{\phi_s}(\cdot) \,\middle\|\, p_{i,t}^{0}(\cdot)\right)\right\rangle_{\mathcal{B}_s^-}^{\mathrm{traj}}, \tag{9}$$

where $\langle\cdot\rangle_{\mathcal{B}_s^-}^{\mathrm{traj}}$ averages over tokens within each trajectory, then over rejected trajectories, as in Eq. (8). We optimize the complete objective:

$$\mathcal{L}_{\mathrm{CoRed}} = \mathcal{L}_{\mathrm{SMO}} + \beta_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}, \tag{10}$$

where $\beta_{\mathrm{KL}} \geq 0$ controls the strength of the KL regularizer. In all experiments, **CoRed (ours)** refers to this combined SMO + KL objective. The reference distribution is computed with the adaptation parameters disabled and remains fixed throughout TTT.

4 Experiments

We conduct experiments to investigate the effectiveness of test-time training under failure-only feedback. We benchmark several existing optimization methods adapted to this setting and compare them with our proposed CoRed method.

Our experiments cover two model families, Qwen3-4B-Instruct (Yang et al., 2025) and Llama-3.2-3B-Instruct (Llama-3 Team, 2024), across three reasoning-intensive domains. We begin with *Countdown*, a controlled arithmetic reasoning task whose trajectories can be readily analyzed to study the dynamics of failure-only TTT (§4.1). We then evaluate on substantially harder mathematical reasoning (§4.2) and competitive programming problems (§4.3).

Methods. We evaluate multiple methods adapted for failure-only TTT as introduced in §2.3, including Naive Sampling, Unlikelihood Training (**UL**) (Welleck et al., 2019), Negative Sample Reinforcement (**NSR**) (Zhu et al., 2025), Entropy Minimization (**EM**) (Gao et al., 2025), and Self-Distillation Policy Optimization (**SDPO**) (Hübotter et al., 2026). We refer to our approach as **CoRed**. Appendix A.2 includes more implementation details and hyper-parameters of our approach and baselines.

Evaluation and Research Questions. We evaluate failure-only TTT along two dimensions: *solvability* and *sample efficiency*. First, we ask *whether TTT can solve problems that remain unsolved by naive sampling under the same inference budget.* Second, for solvable problems, we ask *whether TTT can reduce the number of attempts required to reach the first correct solution*. The former targets extremely hard instances—for example, problems unsolved by the base model after 4,096 rollouts—while the latter measures how efficiently adaptation reaches a solution.

4.1 Illustrative Task: Countdown

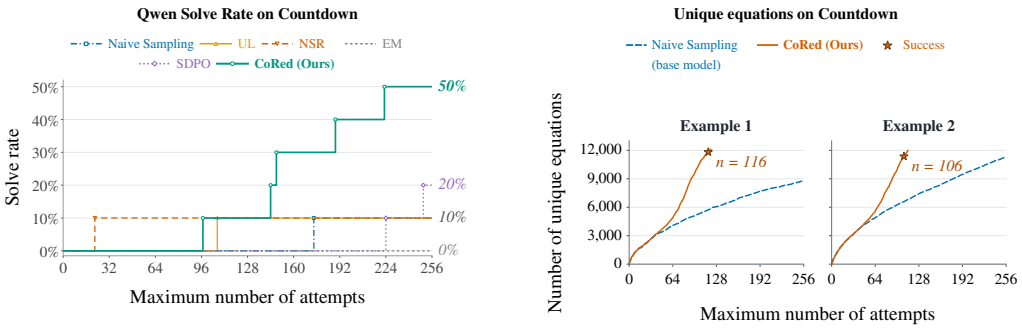

Figure 5: (left) CoRed solves more problems within 256 attempts than all baselines. (right) Number of unique equations in rollouts from countdown problems. Compared with naive sampling, CoRed generates more unique equations and solves more problems which naive sampling cannot solve.

Countdown is an arithmetic reasoning game where a model forms a valid expression from a set of numbers to reach a target value (e.g., composing `40,19,23,7` to get `29` with `40+19-23-7=29`). We use this task for analysis because its rollouts allow easy inspection of intermediate computations.

Setup. We construct a challenging subset of Countdown instances. In this setting, we evaluate only Qwen3-4B-Instruct, as Llama-3.2-3B-Instruct fails completely. To focus on non-trivial cases,

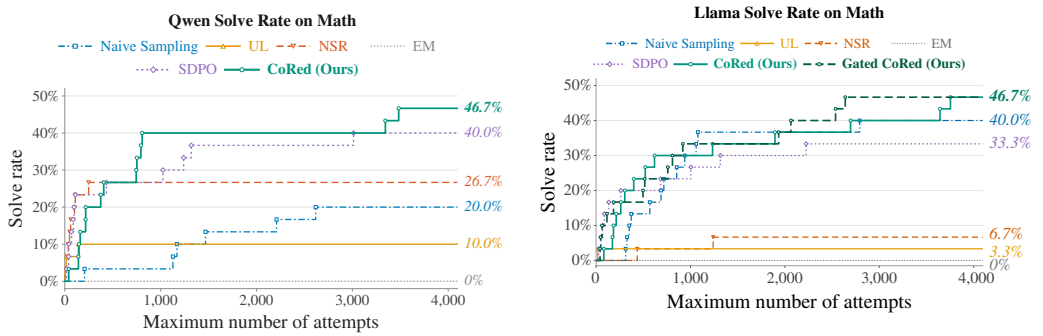

Figure 6: Solve rate on hard math problems as a function of the maximum number of attempts per problem, comparing CoRed with baselines across Qwen3-4B-Instruct and Llama3.2-3B-Instruct.

we select 10 problems that naive sampling fails to solve within 128 attempts, allocating a total budget of 256 attempts per problem. See Appendix A.1 for details

Results. Figure 5 (left) reports the cumulative solve rate (y-axis) as a function of the total number of attempts (x-axis). Compared with naive sampling from the frozen base model, two test-time training methods, CoRed and SDPO, solve more problems under the same attempt budget, indicating that parameter adaptation can improve the effective use of inference-time computation. Among the evaluated methods, CoRed achieves the strongest overall performance.

Test-time Training Encourages Exploration. We analyze how TTT changes rollout behavior by tracking the cumulative number of unique intermediate equations, extracted with regular expressions and deduplicated, across sampled trajectories. Figure 5 (right) shows that CoRed accumulates distinct equations substantially faster than naive sampling from the frozen base model. On the two representative problems, it produces approximately $2\times$ as many unique equations within 128 attempts and finds correct solutions, whereas naive sampling finds none within 256 attempts. This suggests that learning from failures changes subsequent generations to explore a broader and different set of intermediate computations, which may explain the improved sample efficiency of failure-only TTT.

4.2 Results on Math Problems

Setup. We construct model-dependent hard subsets from OlympiadBench (He et al., 2024a), AIME(24-26), and HMMT (Balunović et al., 2025). We evaluate Qwen3-4B-Instruct and Llama-3.2-3B-Instruct, retaining only problems that naive sampling from the corresponding frozen base model fails to solve within 1,024 and 256 attempts, respectively. Due to the computational cost of performing test-time training separately for each problem instance, we subsample each hard subset to 30 problems. We allocate a total budget of 4,096 attempts per problem. Additional details on dataset construction and composition are provided in Appendix A.1.

Results. Figure 6 reports the solve rate as a function of the maximum number of attempts. Overall, the results demonstrate **substantial benefits from test-time training** even when the model learns exclusively from failed rollouts.

Test-time training improves problem solvability. Compared with naive sampling from the frozen base model, several TTT methods solve more problems within the same attempt budget. On Qwen3-4B-Instruct, CoRed, SDPO, and NSR all improve over naive sampling, with CoRed achieving a solve rate of 46.7% compared with 20.0% for the base model. On Llama-3.2-3B-Instruct, CoRed likewise improves over naive sampling, whereas the other baselines fail to surpass it. We observe that naive UL, NSR, and EM often degenerate after many rounds of test-time updates, eventually yielding performance below naive sampling. Overall, CoRed achieves the strongest performance among the evaluated TTT methods.

Failure-only TTT also improves sample efficiency. This advantage is already apparent at substantially smaller attempt budgets. On Qwen3, CoRed solves 40% of the problems within 1,000 attempts, while naive sampling remains below 10%. Across both models, CoRed generally achieves higher solve rates than naive sampling and the strongest competing baseline, SDPO, at comparable attempt budgets. Our primary evaluation measures efficiency in terms of the number of attempts; in §4.4, we further show that these gains in sample efficiency can translate into wall-clock efficiency despite the additional gradient updates required by TTT.

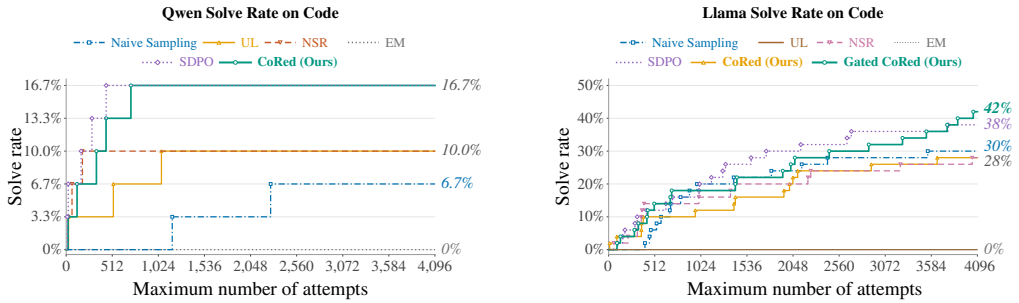

Figure 7: Solve rate on hard code problems as a function of the maximum number of attempts per problem, comparing CoRed with naive sampling and test-time training baselines across two models.

4.3 Results on Code Problems

Setup. We evaluate on hard Codeforces problems from the CodeContests benchmark (Li et al., 2022). We construct model-dependent hard subsets by retaining problems that naive sampling from the corresponding frozen base model fails to solve within 1,024 attempts for Qwen3-4B-Instruct and 256 attempts for Llama-3.2-3B-Instruct. We then subsample 30 problems for Qwen and 50 problems for Llama. Each method is evaluated with a maximum budget of 4,096 attempts per problem. Further details on dataset construction are provided in Appendix A.1.

Results. Figure 7 reports the solve rate as a function of the maximum number of attempts per problem. We observe trends consistent with the math experiments, demonstrating that **failure-only test-time training also provides substantial gains on coding tasks**. Under a budget of 4,096 attempts, TTT methods, especially CoRed and SDPO, solve more problems than naive sampling from the frozen base model. On Qwen3, CoRed solves $2.5\times$ as many problems as naive sampling. We also observe substantial improvements in sample efficiency. At matched solve rates, CoRed requires approximately $8\times$ fewer attempts than naive sampling on Qwen3 and about $2\times$ fewer attempts on Llama-3.2. Overall, the results indicate that failure-only TTT can improve both solve rate and sample efficiency.

4.4 Ablations & Analysis

Ablations: Importance of SMO and KL Constraint. Recall that CoRed integrates SMO to reshape the negative objective and a KL constraint to control distributional drift. We isolate the contribution of each component by evaluating variants of CoRed with SMO or the KL constraint removed.

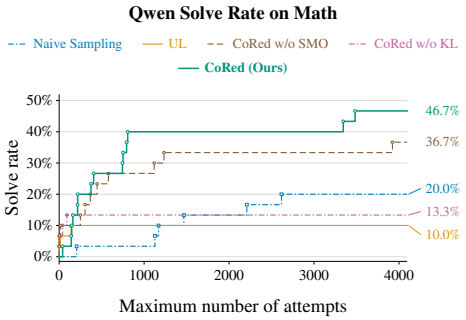

Figure 8: Ablation study. Removing either SMO or KL from CoRed leads to degraded performance.

We report results on math problems with Qwen3-4B-Instruct (see Appendix B.3 for results with Llama-3.2-3B). As shown in Figure 8, **removing either component degrades both final solve rate and sample efficiency**. In particular, removing the KL constraint (CoRed w/o KL) leads to more severe degeneration, indicating its importance for preserving generation quality. Removing SMO (CoRed w/o SMO) consistently weakens performance, suggesting that reshaping the negative penalty improves the effectiveness of failure-only updates. Combining both components yields the strongest and most stable performance.

Analysis: Runtime Efficiency. Our primary metric is the number of attempts, motivated by settings in which evaluation or verification is the main bottleneck. Since TTT additionally incurs the cost of gradient updates, we also examine whether improvements in sample efficiency translate into wall-clock savings. Table 1 reports the average time required for sampling and gradient updates per rollout on math problems. The update overhead is relatively small: approximately 9% of the sampling time for Llama-3.2-3B-Instruct and 26% for Qwen3-4B-Instruct. Consequently, reductions in the number of required attempts can

	Avg. Sampling (s)	Avg. Gradient (s)
Llama-3.2-3B-Ins	5.45	0.48
Qwen3-4B-Ins	12.85	3.35

Table 1: Mean sampling and gradient-update time for CoRed per rollout.

outweigh the additional optimization cost. Figure 9 compares CoRed with naive sampling on the Qwen math problems from §4.2. At a budget of 8 hours per problem, CoRed solves approximately 40% of the problems, compared with less than 20% for naive sampling (see Appendix B.2).

Analysis: Robustness Across Random Seeds. To assess sensitivity to randomness, we reran CoRed with multiple seeds on subsets of the Qwen3-4B-Instruct math and coding problems. The results remained consistent across seeds and did not change our conclusions; see Appendix B.1.

Discussion: Effects of Token Gating. Our experiments show that SMO consistently improves performance across models and tasks by smoothly reshaping negative pressure. On Llama-3.2-3B-Instruct, a more aggressive entropy-gated variant can perform better. We attribute this behavior to the highly skewed token-probability distribution produced by Llama. Figure 10 shows that more than 90% of sampled tokens have probability above 0.9 (see Appendix B.2), so masking updates to low-entropy, high-confidence positions can better concentrate negative updates on uncertain decisions. This result motivates adaptive mechanisms that adjust the strength or form of redistribution to the evolving token-probability or entropy distribution during TTT.

5 Related Work

Test-Time Training. *Test-time scaling* keeps model parameters fixed and spends extra inference compute on repeated sampling, aggregation, verification, or search (Wang et al., 2023b; Yao et al., 2023; Snell et al., 2025). *Test-time training* (TTT) instead adapts the parameters to each target instance. Early TTT methods fine-tune on examples retrieved or selected near the test input (Hardt & Sun, 2024; Hübotter et al., 2025). More recent methods optimize a reward at test time. TTT-Discover (Yuksekgonul et al., 2026) combines an entropic reward objective with candidate reuse. Other works (Surina et al., 2025; Wang et al., 2026) pair parameter updates with evolutionary search and TTRL (Zuo et al., 2025) builds pseudo-rewards from majority voting. Test-time adaptation has produced strong results in autocorrelation inequalities, algorithm design, and GPU kernel optimization (Yuksekgonul et al., 2026; Hubert et al., 2026; Akyürek et al., 2025; Yan et al., 2026; Riaz et al., 2026). Our setting differs in the feedback available during adaptation. Existing instance-specific approaches generally rely on graded scalar rewards to distinguish more promising candidates. We instead study TTT when the verifier provides only binary failure before the first success, with no fine-grained signal indicating which failed trajectory is preferable. This motivates CoRed, which treats adaptation in this regime as controlled redistribution of the model's predictive distribution.

Continual Learning. Our work is also related to continual learning, where models must acquire new information while preserving useful capabilities learned previously. Recent work on LMs studies continual pretraining and knowledge acquisition (Jang et al., 2022; Ke et al., 2023), sequential task and instruction learning (Scialom et al., 2022; Yin et al., 2022), and methods for reducing interference during repeated adaptation. Representative approaches include replay or self-generated rehearsal (Huang et al., 2024; Wang et al., 2024; Feng et al., 2026), selective knowledge distillation (He et al., 2024b), and parameter-efficient methods that isolate or constrain update directions (Wang et al., 2023a; 2025a; Qiao & Mahdavi, 2026; Yang et al., 2026). Recent analyses also connect forgetting to distributional drift from the base model (Zheng et al., 2025; Shenfeld et al., 2026b), motivating self-distillation and on-policy replay methods that better preserve the original policy (Shenfeld et al., 2026a; Chen et al., 2026). While continual learning typically concerns adaptation across a sequence of tasks, our setting performs aggressive adaptation to a *single* test problem; nevertheless, its mechanisms for controlling interference and policy drift are closely related to the regularization used in CoRed.

6 Conclusion

We study test-time training from failures alone, where models receive no fine-grained scalar reward. Across challenging Countdown, math, and coding problems, we benchmark existing approaches and introduce Controlled Redistribution (CoRed), which combines a smoothed negative-learning objective with a base-policy KL constraint. Our results show that failure-only TTT can improve both first-success sample efficiency and problem solvability, including on instances rarely or never solved by repeated sampling from the fixed base model. Analysis on Countdown further suggests that these gains arise in part from reshaping subsequent exploration. Overall, our findings establish failure-only TTT as a viable inference-time adaptation regime.

AI USE STATEMENT

Generative AI tools were used to support improvements in writing quality and to assist with debugging code. The authors carefully reviewed all AI-supported material and remain fully responsible for the final manuscript, its claims, and all associated artifacts.

REFERENCES

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 942–963. PMLR, 2025.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. https://matharena.ai/, February 2025.

Yan Chen, Taojie Zhu, Meng Zhang, Xin Chen, Jiaqi Huang, Dongyang Xu, and Yizhi Wang. On-policy replay for continual supervised fine-tuning. *arXiv preprint arXiv:2605.29495*, 2026.

Yujie Feng, Hao Wang, Jian Li, Xu Chu, Zhaolu Kang, Yiran Liu, Yasha Wang, Philip S. Yu, and Xiao-Ming Wu. FOREVER: Forgetting curve-inspired memory replay for language model continual learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24945–24965. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1144.

Zitian Gao, Lynx Chen, Haoming Luo, Joey Zhou, and Bryan Dai. One-shot entropy minimization, 2025. URL https://arxiv.org/abs/2505.20282.

gauravrghosal. countdown_hard: A hard split of the countdown arithmetic puzzle. Hugging Face dataset, 2026. URL https://huggingface.co/datasets/gauravrghosal/countdown_hard. Revision c879adf.

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

Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *International Conference on Learning Representations*, 2024.

Jan Kristian Haugland. The minimum overlap problem revisited. *arXiv preprint arXiv:1609.08000*, 2016.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems, 2024a. URL https://arxiv.org/abs/2402.14008.

Jinghan He, Haiyun Guo, Kuan Zhu, Zihan Zhao, Ming Tang, and Jinqiao Wang. SEEKR: Selective attention-guided knowledge retention for continual learning of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3254–3266. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.emnlp-main.190.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Jianheng Huang, Leyang Cui, Ante Wang, Chengyi Yang, Xinting Liao, Linfeng Song, Junfeng Yao, and Jinsong Su. Mitigating catastrophic forgetting in large language models with self-synthesized rehearsal. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1416–1428. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.77.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, Amol Mandhane, Jessica Yung, Iuliya Beloshapka, Borja Ibarz, Vivek Veeriah, Lei Yu, Oliver Nash, Paul Lezeau, Salvatore Mercuri, Calle Sönne, Bhavik Mehta, Alex Davies, Daniel Zheng, Fabian Pedregosa, Yin Li, Ingrid von Glehn, Mark Rowland, Samuel Albanie, Ameya Velingker, Simon Schmitt, Edward Lockhart, Edward Hughes, Henryk Michalewski, Nicolas Sonnerat, Demis Hassabis, Pushmeet Kohli, and David Silver. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026. doi: 10.1038/s41586-025-09833-y.

Jonas Hübotter, Sascha Bongni, Ido Hakimi, and Andreas Krause. Efficiently learning at test-time: Active fine-tuning of LLMs. In *International Conference on Learning Representations*, 2025.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation, 2026. URL https://arxiv.org/abs/2601.20802.

Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, Stanley Jungkyu Choi, and Minjoon Seo. Towards continual knowledge learning of language models. In *International Conference on Learning Representations*, 2022.

Zixuan Ke, Yijia Shao, Haowei Lin, Tatsuya Konishi, Gyuhak Kim, and Bing Liu. Continual pre-training of language models. In *International Conference on Learning Representations*, 2023.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Llama-3 Team. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. Tinyzero. https://github.com/Jiayi-Pan/TinyZero, 2025. Accessed: 2025-01-24.

Fuli Qiao and Mehrdad Mahdavi. Merge before forget: A single LoRA continual learning via continual merging. In *International Conference on Learning Representations*, 2026.

Kainat Riaz, Muhammad Ahmed Mohsin, Ahsan Bilal, Muhammad Umer, Ayesha Mohsin, Aqib Riaz, Ali Subhan, and John M. Cioffi. Epistemic uncertainty for test-time discovery, 2026. URL https://arxiv.org/abs/2605.11328.

Thomas Scialom, Tuhin Chakrabarty, and Smaranda Muresan. Fine-tuned language models are continual learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 6107–6122. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.410.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026a.

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. RL's razor: Why online reinforcement learning forgets less. In *International Conference on Learning Representations*, 2026b.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025.

Anja Surina, Amin Mansouri, Lars C. P. M. Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with LLMs: Evolutionary search meets reinforcement learning. In *Conference on Language Modeling*, 2025.

Chenxu Wang, Yilin Lyu, Zicheng Sun, and Liping Jing. Continual gradient low-rank projection fine-tuning for LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14815–14829. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.acl-long.721.

Haozhe Wang, Qixin Xu, Che Liu, Junhong Wu, Fangzhen Lin, and Wenhu Chen. Emergent hierarchical reasoning in llms through reinforcement learning, 2025b. URL https://arxiv.org/abs/2509.03646.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning, 2025c. URL https://arxiv.org/abs/2506.01939.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10658–10671. Association for Computational Linguistics, 2023a. doi: 10.18653/v1/2023.findings-emnlp.715.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023b.

Yifan Wang, Yafei Liu, Chufan Shi, Haoling Li, Chen Chen, Haonan Lu, and Yujiu Yang. InsCL: A data-efficient continual learning paradigm for fine-tuning large language models with instructions. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 663–677. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.naacl-long.37.

Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, Hao Cheng, Pengcheng He, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. ThetaEvolve: Test-time learning on open problems. In *International Conference on Machine Learning*, 2026.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training, 2019. URL https://arxiv.org/abs/1908.04319.

Minghao Yan, Bo Peng, Benjamin Coleman, Ziqi Chen, Zhouhang Xie, Shuo Chen, Zhankui He, Noveen Sachdeva, Weili Wang, Ed H. Chi, Shivaram Venkataraman, Wang-Cheng Kang, Derek Zhiyuan Cheng, and Beidou Wang. Pacevolve++: Improving test-time learning for evolutionary search agents, 2026. URL https://arxiv.org/abs/2605.07039.

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, Jiaxin Yang, Jingren Zhou, Jingren Zhou, Junyan Lin, Kai Dang, Keqin Bao, Ke-Pei Yang, Le Yu, Li-Chun Deng, Mei Li, Min Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shi-Qiang Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *ArXiv*, abs/2505.09388, 2025. URL https://api.semanticscholar.org/CorpusID:278602855.

Lina Yang, Yusheng Liao, Yanfeng Wang, and Yu Wang. SLoRA: Balancing plasticity and forgetting in large language models for continual learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5437–5454. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.247.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Wenpeng Yin, Jia Li, and Caiming Xiong. ConTinTin: Continual learning from task instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3062–3072. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.218.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model?, 2025. URL https://arxiv.org/abs/2504.13837.

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. *arXiv preprint arXiv:2601.16175*, 2026.

Alex L. Zhang, Matej Sirovatka, Erik Schultheis, Benjamin Horowitz, and Mark Saroufim. Kernelbot: A competition platform for writing heterogeneous GPU code. In *Championing Open-source Development in ML Workshop at ICML*, 2025.

Junhao Zheng, Xidi Cai, Shengjie Qiu, and Qianli Ma. Spurious forgetting in continual learning of language models. In *International Conference on Learning Representations*, 2025.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. The surprising effectiveness of negative reinforcement in llm reasoning, 2025. URL https://arxiv.org/abs/2506.01347.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. TTRL: Test-time reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

A EXPERIMENT SETUP

A.1 DATA CONSTRUCTION

We conduct experiments with two instruction-tuned models, Qwen3-4B-Instruct-2507 (Yang et al., 2025) and Llama-3.2-3B-Instruct (Llama-3 Team, 2024), on three domains: the Countdown arithmetic reasoning task, competition-level Math, and competitive Coding. Because we study problems that are hard for the base model, we build a model-specific hard set for each domain. We remove every problem that the base model already solves by naive sampling within our budget. For Math and Code problems, we remove problems solved within 1,024 attempts for Qwen3-4B-Instruct-2507 and 256 attempts for Llama3.2-3B-Instruct. For Countdown, we remove problems solved within 128 attempts by naive sampling from the base model for Qwen3-4B-Instruct-2507.

Attempt budget To keep the comparison fair, every method, including naive sampling, all baselines, and ours, gets the same budget of 4,096 attempts per problem for Math and Code problems and 256 attempts for Countdown problems.

Countdown. We only conduct Countdown experiments on Qwen3-4B-Instruct-2507 because Llama fails at most of problems for Countdown problems with four input numbers. Each Countdown instance gives a set of integers and a target value. The model must write an arithmetic expression that uses each given integer exactly once, combined with the operations $+$, $-$, $\times$, and $\div$, and that evaluates exactly to the target. We verify a candidate by parsing its expression and evaluating it with exact rational arithmetic. We perform sampling on 120 problems in total from Tiny_zero (Pan et al., 2025) and Count_hard (gauravrghosal, 2026). Finally, we select 10 Countdown problems that the base model fails to solve within 128 attempts of naive sampling. Six come from the test split of Countdown_hard (gauravrghosal, 2026): five have six input numbers and one has eight. The other four come from the test splits of the five- and six-number TinyZero Countdown sets (Pan et al., 2025): three have five input numbers and one has six.

Math. For Qwen3-4B-Instruct-2507, we sample 30 challenging problems from OlympiadBench (He et al., 2024a). For Llama-3.2-3B-Instruct, we sample another 30 problems from AIME(24-26), HMMT[1], and OlympiadBench (He et al., 2024a).

Code. We sample 30 problems for Qwen3-4B-Instruct-2507 and 50 problems for Llama-3.2-3B-Instruct from the Codeforces problems in CodeContests (Li et al., 2022). A generated program is counted as correct only if it passes all available test cases for the problem.

A.2 IMPLEMENTATION DETAILS

Objectives and baseline configurations Our method CORED consists of SMO and KL as discussed in §3. For the objective of SMO $-\log(1-p_\theta^\gamma)$, we adopt $\gamma = 0.5$ for all experiments. We use KL_coefficient as 1.5 for Qwen and 3.0 for Llama. For NSR (Zhu et al., 2025), we following their setting to assign failed attempts reward -1. For SDPO, we follow Hübotter et al. (2026) to minimize reverse KL to a stop-gradient, feedback-conditioned teacher with EMA momentum 0.99. For all of tasks, we only apply wrong-answer verdict feedback to teacher to implement SDPO.

Adapter and optimization. For each problem, we initialize a fresh LoRA adapter (Hu et al., 2021) and keep all pretrained weights frozen. We apply LoRA to the query, key, value, and output projections ($q/k/v/o$) of every attention layer, with rank $r = 32$ and scaling factor $\alpha = 64$. We optimize the adapter with AdamW, using a learning rate of 1×10^{-4} and clipping the gradient norm at 1.0.

Adaptation loop. Adaptation proceeds in iterations. At each iteration, we sample $k = 8$ candidate solutions from the current adapted model and verify each one. If any candidate is correct, adaptation stops and the problem is marked as solved. Otherwise, all k candidates have failed, and we take a single optimizer step on this batch of failed attempts. The loop repeats until a verified solution is

[1] https://www.hmmt.org

found or the per-problem budget of 4,096 attempts is used up. With $k = 8$, this budget allows at most 512 adaptation steps.

Decoding. We follow the recommended decoding setting for corresponding models, following Yue et al. (2025). Specifically, we use temperature of $T = 0.6$, top-p value of 0.95, allowing a maximum generation of 16,384 tokens. We apply this setting to all baselines and our methods. Prompts are formatted with each model's chat template.

B Additional Results

B.1 Multi-Seed Robustness

PID	Naive Sampling Corr./N	CoRed (ours) # Attempts	Avg. # Attempts
2111	1/1128	146, 116, 163	141.67
2153	1/1206	100, 209, 173	160.67
2255	0/4096	37, 53, 20	36.67
2384	0/4096	166, 333, 152	217.00
2919	2/2062	14, 340, 300	218.00

Table 2: Multi-seed CoRed (ours) results on randomly selected math problems. Lower # Attempts is better.

PID	Naive Sampling Corr./N	CoRed (ours) # Attempts	Avg. # Attempts
1515_D	7/9280	19, 191, 44, 228	120.50
1796_C	1/8192	79, 569, 481, 642	442.75
329_A	11/8192	47, 10, 10, 15	20.50
819_C	1/8192	227, 460, 646, 11	336.00

Table 3: Multi-seed CoRed (ours) results on randomly selected Codeforces problems. Lower # Attempts is better.

Our main experiments run each TTT method once per problem due to the substantial computational cost of per-problem TTT. To assess the robustness of the observed gains, we randomly select subsets of math and coding problems and rerun CoRed with multiple random seeds on Qwen3-4B-Instruct, resetting the test-time adapter before each run. We do not repeat naive sampling experiments, since individual samples are drawn independently from the frozen model, whereas TTT trajectories are path-dependent through sequential parameter updates.

Tables 2 and 3 report multi-seed robustness. For each problem and seed, we reset the test-time adapter and rerun CoRed (ours) under the same failure-only protocol as in the main experiments. Each value in the CoRed (ours) # Attempts column is one seed's first-success time $T^\star(x)$, and Avg. # Attempts is the mean over the listed seeds. In Table 3, Naive Sampling Corr./N reports the number of correct solutions found by naive sampling from the frozen base model among N rollouts. Across different problems, CoRed finds a correct solution in every reported run with comparable number of attempts, suggesting that the observed improvements are not driven by a single favorable optimization trajectory.

B.2 Additional Analysis

Solve rate over estimated time. Figure 9 complements the runtime analysis in §4.4. Estimated time accounts for sampling and gradient updates using the mean latencies in Table 1. Despite the update overhead, CoRed retains a substantial advantage over naive sampling: at eight hours per problem, it solves approximately 40% of the Qwen math problems, compared with less than 20% for naive sampling. Thus, the reduction in required attempts can translate into time savings.

Token probability histogram. Figure 10 compares sampled-token probabilities for Qwen3-4B-Instruct and Llama-3.2-3B-Instruct. Llama's distribution is more concentrated near one, with roughly

90% of tokens in the highest-probability bin. This motivates the token-gating discussion in §4.4: when highly confident tokens dominate a trajectory, restricting negative updates to high-entropy positions may help focus adaptation on uncertain decisions.

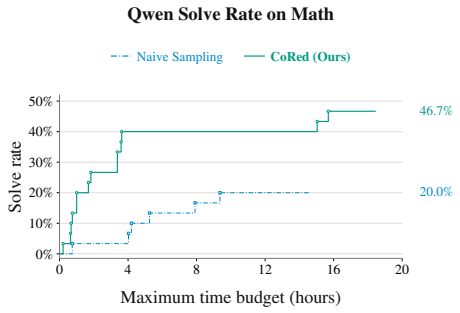

Figure 9: Solve rate on Qwen math problems versus estimated time, including sampling and gradient-update costs.

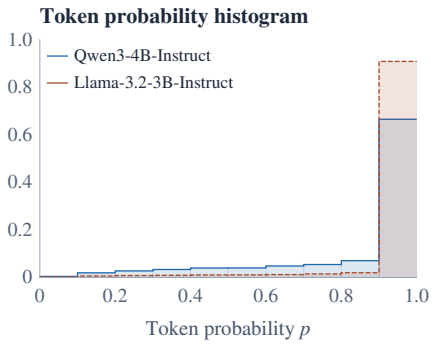

Figure 10: Sampled-token probability distributions. Llama assigns a larger fraction of tokens probabilities near one than Qwen.

B.3 Ablations on Llama

Figure 11 reports the corresponding math ablations on Llama-3.2-3B-Instruct. We observe similar trends to Qwen: both SMO and the KL constraint contribute to performance, and removing the KL constraint causes the largest degradation.

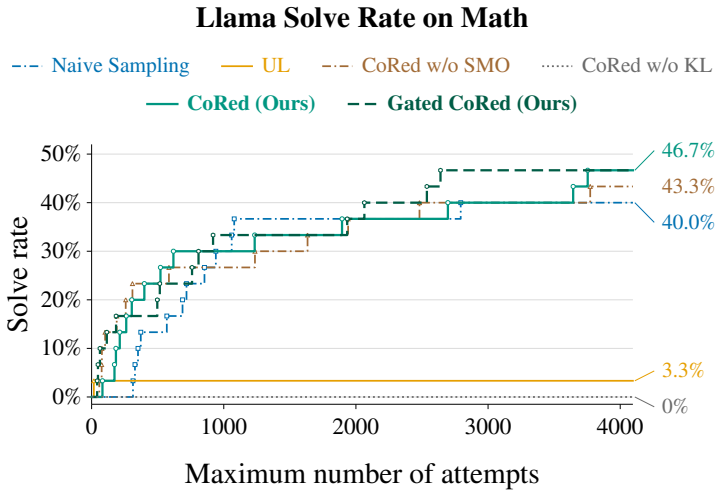

Figure 11: Math ablations on Llama-3.2-3B-Instruct. CoRed combines SMO with the base-policy KL constraint; Here we include the variants of CoRed w/o SMO and KL separately. The gated variant, naive sampling, and UL are included for comparison.

B.4 Gradient Effects of Negative Sample Reinforcement

We briefly examine NSR in the same manner as UL in §3. Let $p = \mathrm{softmax}(z)$ denote the next-token distribution at a fixed prefix and y the sampled token in a rejected trajectory. Following the token-level gradient analysis of Zhu et al. (2025), for the per-token loss $\ell_{\mathrm{NSR}} = p_y$ used in §2.3, the gradient is

$$\frac{\partial \ell_{\mathrm{NSR}}}{\partial z_y} = p_y(1 - p_y). \tag{11}$$

Thus, NSR also suppresses the sampled logit, but its negative pressure is strongest at intermediate probabilities and vanishes at both extremes. Compared with UL, the additional factor $(1 - p_y)$ reduces suppression of highly confident tokens. However, low-probability choices still receive weak updates, including potentially important reasoning branches. This may potentially contribute to NSR's weaker performance relative to CoRed in §4, as CoREDexplicitly amplifies negative pressure on low-probability sampled tokens through SMO or gating.