TEST-TIME LEARNING FROM REASONING FAILURES

Anonymous authors
Paper under double-blind review

ABSTRACT

When an attempt is rejected during deployment, large language models (LLMs) commonly retry through resampling or textual feedback for self-correction, while keeping parameters fixed. Yet a rejected attempt itself provides a negative learning signal: the model can update its parameters to reduce the attempt's likelihood, without adding it to the context. However, simply unlearning a rejected attempt before retrying does not guarantee better overall performance. To this end, we introduce NEGATIVE SPACE, a bilevel optimization framework that trains LLMs to adapt from rejected attempts at test time. Specifically, during training, the lower level simulates test-time adaptation from failure: it applies a single unlearning step to rejected responses between two successive attempts. The upper level learns to make this adaptation beneficial for overall performance through first-order policy optimization that maximizes rewards across initial and retry attempts. At test time, a rejected attempt triggers the same unlearning step; the model then retries the original query and discards the update. Across code generation and mathematical reasoning, NEGATIVE SPACE improves average accuracy over the strongest evaluated retry baseline by 4.8 points on two coding benchmarks and 3.7 points on five mathematics benchmarks. Further analysis shows that, although trained with fewer retries, it retains its advantage when given more at test time.

1 INTRODUCTION

Large language models (LLMs) often need multiple attempts to solve problems in code generation and math reasoning. When a generated program fails a test, for example, the system can sample another solution or ask the model to revise its code using the test feedback (Kumar et al., 2024). These approaches differ in what the model sees on its next attempt: independent retry uses the original query, while self-correction also includes the previous response and feedback (Figure 1a,b). In either case, the model's parameters remain unchanged. In this paper, we ask: *Can a failed attempt instead guide a parameter update that improves the next response?*

A failed attempt still offers a learning signal. Prior work showed that penalizing incorrect responses can improve reasoning during RLVR training (Zhu et al., 2025). At test time, a similar signal can drive an unlearning step that lowers the likelihood of a rejected response before the model retries. However, suppressing an entire response may also penalize useful intermediate steps. Our ablation shows that such updates can hurt retry performance (Table 3a), thus motivating learning how to turn reasoning failures into updates that improve subsequent attempts.

We introduce NEGATIVE SPACE, a bilevel optimization framework that meta-learns how to adapt from failures at test time. During training, the two levels connect local unlearning within each query to the objective of improving overall performance: *the lower level suppresses rejected responses, while the upper level learns how to make these updates useful for the next attempt*. At inference, the model adapts to unlearn each new failed attempt in the parameter space and retries the original query without having the rejected response in the context (Figure 1c).

Concretely, NEGATIVE SPACE separates what the model learns across queries from how it adapts to a particular query. It uses an asymmetric LoRA adapter (Hu et al., 2022): the shared factor $\mathbf{A}$ learns how to make unlearning useful, while the query-specific factor $\mathbf{B}$ updates the model weights to learn from the failure. During training, the model updates $\mathbf{B}$ to reduce the likelihood of a rejected response, then retries the original query. Rewards from both attempts train the shared policy and $\mathbf{A}$, encouraging updates that improve subsequent responses. At test time, the shared parameters remain fixed, and a fresh $\mathbf{B}$ is adapted for each query's reasoning failures.

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

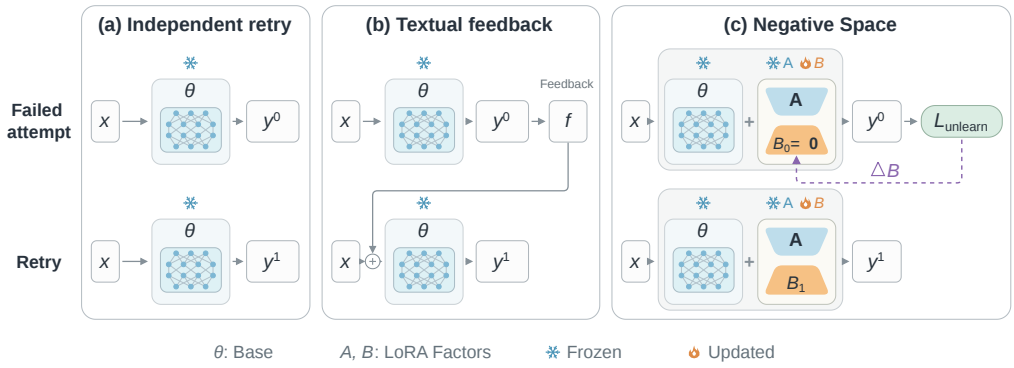


Figure 1: Comparison of three test-time retry paradigms. Given a query x, a rejected initial response y^0 ($V(x, y^0) = 0$) triggers a retry y^1. (a) Independent retry resamples from the unchanged base policy. (b) Self-correction adds y^0 and textual feedback f to the context; the circled plus denotes concatenation. (c) NEGATIVE SPACE uses y^0 for one unlearning step on the query-specific factor $\mathbf{B}$, updating $\mathbf{B}_0 = \mathbf{0}$ to $\mathbf{B}_1$ while keeping θ and $\mathbf{A}$ fixed. The retry uses the original query without failure text and $\mathbf{B}$ is reset before each new query.

We evaluate NEGATIVE SPACE on code generation and mathematical reasoning, using public tests and majority voting, respectively, to identify responses for adaptation. Our method outperforms the strongest evaluated retry-trained baseline in each domain, improving accuracy over independent-retry training by 5.4 percentage points on CodeContests and 4.1 points on LiveCodeBench, and raising the five-benchmark mathematics average over self-correction training by 3.7 points. Further analysis shows that its retries make larger structural changes to rejected programs and repair a higher fraction of failures than either baseline. Our method generalizes beyond its training retry budget. Although trained with a single retry, its performance scales monotonically and far more effectively than independent retry when the number of test-time retries increases.

2 PROBLEM SETUP: MULTI-ATTEMPT REASONING

We consider reasoning tasks in which a language model can make multiple attempts at the same query. Let π_θ denote the model, x a query, and y a complete response. Queries are split into a training set $\mathcal{D}_{\text{train}}$ and a disjoint test set $\mathcal{D}_{\text{test}}$. We first define the two signals involved, then the retry protocol and its evaluation.

Acceptance check and task score. Whether to retry must be decided without knowing the correct answer. We therefore separate an *acceptance check* $V(x, y) \in \{0, 1\}$, available at test time, from a *task score* $r(x, y) \in \{0, 1\}$, which provides rewards on training queries and grades returned responses on test queries but is never available at test time. A response is *rejected* if $V(x, y) = 0$. In our two settings, the signals are defined as follows:

	Code generation	**Mathematical reasoning**
$V(x, y) = 1$ if y	passes the public tests	matches the majority answer
$r(x, y) = 1$ if y	passes the hidden tests	matches the reference answer

The majority answer is taken over sampled solutions (Zuo et al., 2025). The check thus only approximates the score: public tests can miss bugs, and the majority can be wrong.

Retry protocol and evaluation. Given a budget of $K \geq 0$ retries, the model samples a first attempt $y^0 \sim \pi_\theta(\cdot \mid x)$ and, after each rejection, generates the next attempt with a retry method. Retry methods differ in what changes between attempts (Figure 1):

$$y^{k+1} \sim \begin{cases} \pi_\theta(\cdot \mid x), & \text{independent retry,} \\ \pi_\theta(\cdot \mid x \oplus y^k \oplus f^k), & \text{retry with feedback,} \\ \pi_{\theta+\Delta_{k+1}}(\cdot \mid x), & \text{parameter adaptation,} \end{cases} \tag{1}$$

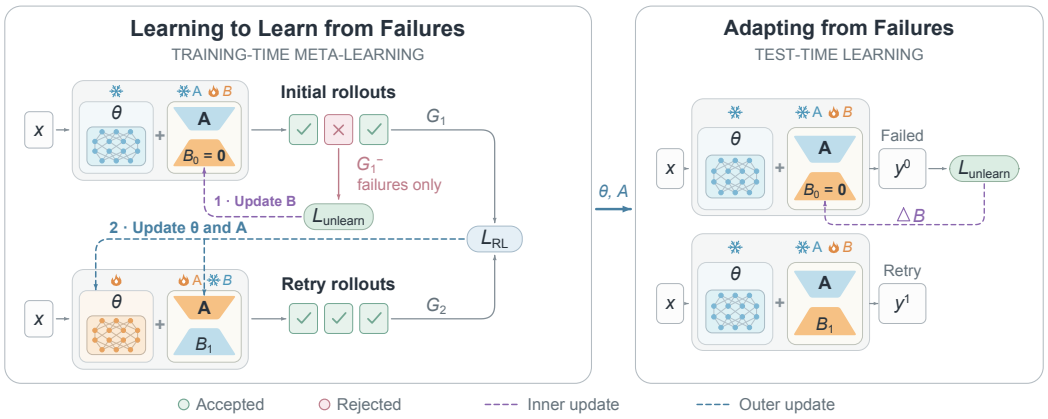


Figure 2: **NEGATIVE SPACE as bilevel learning from failures. (a) Training.** For each query, the lower level takes one unlearning step from $\mathbf{B}_0 = \mathbf{0}$ to $\mathbf{B}_1$ using the rejected responses of the initial group, with θ and $\mathbf{A}$ fixed. The adapted policy then samples a retry group. The upper level updates θ and $\mathbf{A}$ using task rewards from both groups, holding $\mathbf{B}_1$ fixed. **(b) Deployment.** With θ and $\mathbf{A}$ frozen, a rejected attempt triggers an update to $\mathbf{B}$ and a retry on the original query without failure text. $\mathbf{B}$ is reset for each query. Snowflakes and flames mark frozen and updated parameters; dashed arrows denote parameter updates. Green and pink cards illustrate acceptance and rejection under V.

where f^k is textual feedback on y^k, $\oplus$ denotes concatenation, and Δ_{k+1} is a temporary parameter change computed from the rejected attempts. The first two methods keep the parameters fixed and change only the prompt; ours keeps the prompt and changes the parameters. The system returns

$$y^{\text{out}} = \begin{cases} y^k, & \text{if } y^k \text{ is the first accepted attempt,} \\ y^K, & \text{if no attempt is accepted.} \end{cases} \tag{2}$$

We report deployed accuracy, the task score of the returned response on held-out queries:

$$\mathbb{E}_{x \sim \mathcal{D}_{\text{test}}} \mathbb{E}\big[r(x, y^{\text{out}})\big], \tag{3}$$

With $K = 0$, this reduces to first-attempt accuracy. In the method section, $\mathcal{D}$ abbreviates $\mathcal{D}_{\text{train}}$.

3 METHOD

NEGATIVE SPACE learns across training queries how to adapt from rejected attempts on a new query. We first model the relationship between query-specific unlearning and the multi-attempt objective through a bilevel formulation (Section 3.1). We then derive the local unlearning update and explain how the upper level shapes its effect on the model through first-order policy optimization (Section 3.2). Fig. 2 illustrates training and deployment, and Alg. 1 summarizes training.

3.1 BILEVEL LEARNING FROM REJECTED ATTEMPTS

We use LoRA (Hu et al., 2022) to separate learning across training queries from adaptation within each query. Let θ denote all parameters of the base model π_θ. For each weight matrix W selected for adaptation, LoRA adds a rank-d correction,

$$W \;\mapsto\; W + \mathbf{B}\mathbf{A}, \tag{4}$$

where we absorb the LoRA scaling factor into $\mathbf{A}$. We write $\pi_{\theta,\mathbf{A},\mathbf{B}}$ for the adapted policy, using $\mathbf{A}$ and $\mathbf{B}$ collectively for the factors of all adapted matrices.

Our design assigns the two factors different roles. During training, θ and $\mathbf{A}$ are learned across queries. Within each query, we hold them fixed and update only $\mathbf{B}$ using the rejected responses. At deployment, the learned θ and $\mathbf{A}$ remain fixed, and only $\mathbf{B}$ is adapted; the temporary change Δ_k in equation 1 is thus $\mathbf{B}_k\mathbf{A}$ in each adapted matrix. We initialize $\mathbf{A}$ to a nonzero matrix at the start of training and reset $\mathbf{B}_0 = 0$ for every query during both training and deployment. Thus each query starts from the base policy, $\pi_{\theta,\mathbf{A},0} = \pi_\theta$, and updates to $\mathbf{B}$ never accumulate across queries.

Unlearning objective. Section 2 defines the expected task score of the returned response as the system objective. We now specify how rejected responses determine a local parameter update. Let $\mathcal{N}_0$ be the set of rejected responses: $\{y^0\}$ if the first attempt $y^0 \sim \pi_\theta(\cdot \mid x)$ is rejected, and empty otherwise. Holding θ and $\mathbf{A}$ fixed, we define

$$\mathcal{L}_{\text{unlearn}}(\mathbf{B}) = \frac{1}{|\mathcal{N}_0|} \sum_{y \in \mathcal{N}_0} \frac{1}{|y|} \sum_{t=1}^{|y|} \log \pi_{\theta,\mathbf{A},\mathbf{B}}(y_t \mid x, y_{<t}), \tag{5}$$

where $|y|$ is the response length in tokens, and we set the loss to zero when $\mathcal{N}_0$ is empty. Minimizing the loss suppresses rejected responses.

Bilevel formulation. We model the relationship between the system's overall objective and query-specific unlearning through bilevel optimization. The upper level learns θ and $\mathbf{A}$ to maximize the expected task score of the returned response, while the lower level determines the local factor $\mathbf{B}$ from rejected responses. We consider an initial attempt followed by at most one retry ($K = 1$), giving the following formulation:

$$\begin{aligned} \max_{\theta,\mathbf{A}} \quad & \mathbb{E}_{\substack{x \sim \mathcal{D}_{\text{train}} \\ y^0 \sim \pi_\theta(\cdot|x)}} \Big[\underbrace{V(x, y^0)\, r(x, y^0)}_{\text{reward of an accepted first attempt}} \\ & \underbrace{+ [1 - V(x, y^0)]\, \mathbb{E}_{y^1 \sim \pi_{\theta,\mathbf{A},\mathbf{B}^\star}(\cdot|x)}[r(x, y^1)]}_{\text{reward of a retry after rejection}} \Big], \\ \text{s.t.} \quad & \mathbf{B}^\star(\theta, \mathbf{A}) = \arg\min_{\mathbf{B}} \Big\{ \underbrace{\langle \nabla_{\mathbf{B}} \mathcal{L}_{\text{unlearn}}(0), \mathbf{B} \rangle_F}_{\text{linearized unlearning loss}} + \underbrace{\frac{\|\mathbf{B}\|_F^2}{2\eta}}_{\text{update-size penalty}} \Big\}. \end{aligned} \tag{6}$$

Here $\eta > 0$ is the adaptation step size. The lower-level constraint specifies one unlearning step from $\mathbf{B}_0 = 0$ through a regularized linear subproblem: the linear term captures the first-order change in $\mathcal{L}_{\text{unlearn}}$, the quadratic term controls the update magnitude, and the unique minimizer is the gradient step derived in Section 3.2.1.

This formulation has a nested structure: the upper-level objective depends on the lower-level solution $\mathbf{B}^\star(\theta, \mathbf{A})$, because retries are sampled from the adapted policy $\pi_{\theta,\mathbf{A},\mathbf{B}^\star}$. Intuitively, *the lower level uses the rejected responses to specify a local parameter change, while the upper level optimizes overall system performance given this change*. The shared parameters are therefore trained in anticipation of the adaptation performed at deployment.

The formulation 6 extends naturally to larger budgets: each further rejection triggers the same unlearning step, starting from the current $\mathbf{B}$, and the upper objective still scores the returned response.

3.2 Learning the Adaptation Geometry via First-Order Optimization

We first derive the lower-level solution and explain how $\mathbf{A}$ shapes the adaptation geometry—the directions and scale of the resulting weight update. We then describe how task rewards train the shared parameters through first-order policy optimization.

3.2.1 Local Unlearning and Adaptation Geometry

For fixed θ and $\mathbf{A}$, the optimality condition of the lower-level problem in equation 6 is $\nabla_{\mathbf{B}} \mathcal{L}_{\text{unlearn}}(0) + \mathbf{B}/\eta = 0$, which gives the one-step update

$$\mathbf{B}_1 = -\eta\, \nabla_{\mathbf{B}} \mathcal{L}_{\text{unlearn}}(0). \tag{7}$$

Computing it requires no inner optimization loop, only one forward and backward pass over the rejected responses.

To see how $\mathbf{A}$ shapes this step, consider one adapted weight matrix W and let $G = \nabla_W \mathcal{L}_{\text{unlearn}}$ be the unlearning gradient with respect to W at $\mathbf{B}_0 = 0$. The chain rule gives

$$\nabla_{\mathbf{B}} \mathcal{L}_{\text{unlearn}}(0) = G\mathbf{A}^\top, \qquad \Delta W = \mathbf{B}_1 \mathbf{A} = -\eta\, G \underbrace{\mathbf{A}^\top \mathbf{A}}_{\text{learned preconditioner}}. \tag{8}$$

Although $\mathbf{A}$ has no effect on the first attempt, it enters the adaptation step twice: through $G\mathbf{A}^\top$ when computing the gradient for $\mathbf{B}$, and through $\mathbf{B}_1\mathbf{A}$ when mapping the local update to the model's weights. Their combined effect, $\mathbf{A}^\top\mathbf{A}$, is a positive-semidefinite right preconditioner of rank at most d. *At the lower level, rejected responses provide the unlearning gradient G, while the upper level learns $\mathbf{A}$ to control the directions and strength of the resulting weight update.*

Note that the same geometry also applies to multiple retries ($K > 1$). With θ and $\mathbf{A}$ fixed, k unlearning steps from $\mathbf{B}_0 = 0$ give the cumulative weight change

$$\Delta W_k = \mathbf{B}_k\mathbf{A} = -\eta \left(\sum_{\ell=0}^{k-1} G_\ell \right) \mathbf{A}^\top\mathbf{A}, \tag{9}$$

where G_ℓ is the unlearning gradient for the response rejected at round ℓ, evaluated under the adapted policy $\pi_{\theta,\mathbf{A},\mathbf{B}_\ell}$. Every retry thus acts through the same learned preconditioner, so the geometry learned with one retry applies unchanged to larger budgets, as confirmed in Fig. 3a. Section 5 examines the roles of the learned geometry and query-specific adaptation through component ablations.

3.2.2 Learning Shared Parameters from Task Rewards

As in equation 6, the upper level learns θ and $\mathbf{A}$ from task rewards on both initial attempts and adapted retries. We optimize it with a first-order policy-optimization surrogate that holds the computed local factor fixed, avoiding second-order differentiation through the unlearning step.

Specifically, for each training query (Figure 2a), we first sample a group $\mathcal{Y}^0$ of initial responses from $\pi_\theta(\cdot \mid x)$ and collect the rejected ones in $\mathcal{N}_0 \subseteq \mathcal{Y}^0$. If $\mathcal{N}_0$ is nonempty, we perform the unlearning step: we compute $\mathbf{B}_1$ from all responses in $\mathcal{N}_0$ by equation 7 and sample a group $\mathcal{Y}^1$ of retries from the adapted policy $\pi_{\theta,\mathbf{A},\mathbf{B}_1}(\cdot \mid x)$. All responses in both groups are then scored by r, and each group provides one term of the bilevel objective in equation 6. Applying the GRPO loss (Shao et al., 2024) to each group gives the upper-level training loss

$$\mathcal{L}_{\mathrm{RL}}(\theta, \mathbf{A}) = \mathcal{L}_{\mathrm{GRPO}}(\mathcal{Y}^0; \theta) + \left(1 - \bar{V}\right) \mathcal{L}_{\mathrm{GRPO}}(\mathcal{Y}^1; \theta, \mathbf{A} \mid \mathbf{B}_1), \tag{10}$$

where $\mathcal{L}_{\mathrm{GRPO}}(\mathcal{Y}; \cdot)$ is the standard GRPO objective on group $\mathcal{Y}$ with advantages standardized within the group, and $\mathbf{B}_1$ is held fixed. Here $\bar{V}$ is the fraction of initial responses accepted by V, so $1 - \bar{V}$ is the sample counterpart of the gate $1 - V(x, y^0)$ that weights the retry reward in equation 6; the retry term is dropped when $\mathcal{N}_0$ is empty. Because $\mathbf{A}$ has no effect on the policy at $\mathbf{B} = 0$, the first term updates only θ, whereas the second updates both θ and $\mathbf{A}$.

With $\mathbf{B}_1$ held fixed, the retry reward reaches $\mathbf{A}$ through the adapted weights $W' = W + \mathbf{B}_1\mathbf{A}$. For one adapted matrix, the chain rule gives

$$\nabla_{\mathbf{A}} \mathcal{L}_{\mathrm{RL}} = \mathbf{B}_1^\top H, \qquad H = \nabla_{W'} \Big[\left(1 - \bar{V}\right) \mathcal{L}_{\mathrm{GRPO}}(\mathcal{Y}^1; \theta, \mathbf{A} \mid \mathbf{B}_1) \Big]. \tag{11}$$

The retry reward thus trains $\mathbf{A}$ to turn the failure-driven update $\mathbf{B}_1$ into a weight change that lowers the retry loss, and the updated $\mathbf{A}$ shapes unlearning on the next query through equation 8.

Deployment. At test time, θ and $\mathbf{A}$ are frozen: each rejection triggers one unlearning step from the current $\mathbf{B}$ on the newly rejected response and a retry on the original query, following equation 2, and $\mathbf{B}$ is reset before the next query (Figure 2b).

4 Experiments

4.1 Setup

Models and data. We use the post-trained Qwen3-4B and 8B models (Yang et al., 2025) in non-thinking mode for code generation and mathematical reasoning, respectively. For code generation, we use a training pool of 7,054 problems from CodeContests (Li et al., 2022). For mathematics, we follow SimpleRL (Zeng et al., 2025) and train on the hard subset of MATH (Hendrycks et al., 2021), comprising approximately 8.5K problems at difficulty levels 3–5. We evaluate code generation on held-out CodeContests problems and LiveCodeBench (Jain et al., 2025), and mathematical reasoning on MATH-500 (Lightman et al., 2024), AIME 2024, AMC 2023, OlympiadBench (He et al.,

Algorithm 1 NEGATIVE SPACE: first-order training for one-step test-time adaptation

Require: training queries $\mathcal{D}_{\text{train}}$; base parameters θ; shared factor $\mathbf{A}$ (nonzero initialization); step size η

for each training iteration **do**
  Sample a batch of queries from $\mathcal{D}_{\text{train}}$
  for each query x in the batch **do**
    Sample a group $\mathcal{Y}^0$ from $\pi_\theta(\cdot \mid x)$ ▷ $\mathbf{B}_0 = 0$
    $\mathcal{N}_0 \leftarrow \{y \in \mathcal{Y}^0 : V(x, y) = 0\}$; $\bar{V} \leftarrow 1 - |\mathcal{N}_0|/|\mathcal{Y}^0|$
    if $\mathcal{N}_0 \neq \emptyset$ **then**
      Compute $\mathbf{B}_1$ from $\mathcal{N}_0$ by equation 7 ▷ lower level; $\theta, \mathbf{A}$ fixed
      Sample a retry group $\mathcal{Y}^1$ from $\pi_{\theta,\mathbf{A},\mathbf{B}_1}(\cdot \mid x)$
    end if
    Score all responses with r; store their sampling-time log-probabilities
  end for
  Update $\theta, \mathbf{A}$ by minimizing $\mathcal{L}_{\text{RL}}$ in equation 10 ▷ upper level; $\mathbf{B}_1$ held fixed
  Discard all $\mathbf{B}_1$
end for
return $\theta, \mathbf{A}$

Table 1: Code generation: deployed accuracy (%) with Qwen3-4B. Base/GRPO rows use one attempt; retry rows allow one retry. Each of the first two blocks shares a policy; final-block policies are trained separately with their respective retry mechanisms.

Method	CodeContests	LiveCodeBench	AVG
No RL training			
Qwen3-4B	10.7	24.6	17.6
+ independent retry	14.1	29.2	21.7
+ retry with feedback	14.1	26.3	20.2
RL training without a retry loop			
GRPO	15.0	33.0	24.0
+ independent retry	24.3	38.3	31.3
+ retry with feedback	20.9	34.5	27.7
RL training with a retry loop			
Independent retry	25.7	38.3	32.0
Retry with feedback	21.8	35.7	28.8
NEGATIVE SPACE	**31.1**	**42.4**	**36.8**

2024), and HMMT 2025 (Balunović et al., 2025). We report deployed accuracy defined in Eq.3. All main results are averaged over two seeds.

Baselines. All methods use the same backbone within each domain. Retry methods use the same acceptance check and retry budget. (i) **Released model**: The Qwen3 checkpoint without task-specific training. (ii) **GRPO / TTRL**: RL without retry training. For code, GRPO (Shao et al., 2024) uses individual-response task rewards on the offline training set. For mathematics, TTRL (Zuo et al., 2025) updates the model on unlabeled queries using majority-vote rewards, without reference answers. (iii) **Independent retry**: After rejection, the model samples again from the original query (Brown et al., 2024). (iv) **Self-correction**: The retry prompt includes the rejected response and feedback. Code follows Self-Debugging (Chen et al., 2024); mathematics uses majority-vote disagreement as feedback. We apply (iii) and (iv) to both the released model and its RL baseline, with fixed parameters between attempts. (v) **RL with independent retry** and (vi) **RL with self-correction**: Separate policies trained with GRPO rollouts that include the respective retry. Both use rewards from initial and retry responses, our GRPO pipeline, and the same offline training data as NEGATIVE SPACE. Method (v) follows the motivation of inference-aware fine-tuning (Chow et al., 2025); method (vi) adapts RLEF's feedback-conditioned training (Gehring et al., 2025). See implementation details in Appendix A.

Table 2: Mathematical reasoning: deployed accuracy (%) with Qwen3-8B.

Method	AIME	AMC	OlympiadBench	MATH-500	HMMT	AVG
No RL training						
Qwen3-8B	23.3	65.0	53.6	83.2	10.0	47.0
+ independent retry	33.3	67.5	56.1	86.4	10.0	50.7
+ retry with feedback	26.7	65.0	54.4	84.8	13.3	48.8
RL training without a retry loop						
TTRL	33.3	70.0	57.8	87.8	10.0	51.8
+ independent retry	40.0	72.5	60.1	89.6	10.0	54.4
+ retry with feedback	33.3	70.0	60.0	88.6	10.0	52.4
RL training with a retry loop						
Independent retry	30.0	80.0	61.0	87.4	10.0	53.7
Retry with feedback	40.0	77.5	59.9	89.6	13.3	56.1
NEGATIVE SPACE	**43.3**	**80.0**	**62.1**	**90.4**	**23.3**	**59.8**

4.2 MAIN RESULTS

NEGATIVE SPACE outperforms all baselines for both coding and math Our method achieves the highest average deployed accuracy in both domains (Tables 1 and 2). Compared with the strongest evaluated retry baseline in each domain, it improves average accuracy by 4.8 percentage points on code generation and 3.7 points on mathematical reasoning under the same retry budget. It achieves the highest accuracy on both code benchmarks and the highest or tied-highest accuracy on all five mathematics benchmarks.

Training with retries alone yields limited gains. Adding retries at inference improves accuracy on average for both released and RL-trained models, but training through the retry loop provides little consistent additional benefit. For code, both retry-trained baselines only slightly improve over their inference-time counterparts. For math, retry with feedback benefits from training, whereas the trained independent-retry underperforms applying that to TTRL. All retry-trained methods use the same GRPO pipeline and offline data, yet NEGATIVE SPACE outperforms both retry-trained baselines, showing that adapting parameters after a failure is more effective than training a fixed policy to retry.

NEGATIVE SPACE succeeds across domains with different retry preferences. The strongest baseline retry mechanism changes across domains. In code, independent retry consistently outperforms feedback-based retry. Figure 4 provides a potential explanation: feedback-conditioned retries make the smallest edits and repair the fewest failures, suggesting that they remain too close to the rejected program. In mathematics, independent retry is stronger without retry training, but feedback-based retry becomes stronger after training. NEGATIVE SPACE achieves higher average accuracy than both mechanisms across domains. The gains in mathematics also show that NEGATIVE SPACE remains effective when consensus feedback is imperfect.

NEGATIVE SPACE generalizes out-of-distribution and improves where other methods stagnate. In code, our method retains its advantage on LiveCodeBench despite problems shifting to other platforms and later contests, suggesting its gains extend beyond the training domain. In mathematics, although its margin over the strongest baseline is narrow on the in-domain MATH-500, it is much stronger on OOD sets. HMMT provides the clearest contrast, where most baselines remain at the released model's accuracy, even with an RL-trained retry loop, whereas NEGATIVE SPACE more than doubles the scores. Together, we show that learning from failed attempts is particularly useful on difficult problems where conventional training and retry strategies offer limited gains.

5 ANALYSIS AND DISCUSSION

We examine scaling with more retries and independent deployment runs, characterize retry behavior, and then ablate component contributions and analyze test-time efficiency.

Generalization to larger retry budgets. Although trained with one retry, NEGATIVE SPACE continues to improve as the test-time compute budget increases (Figure 3a). One retry already matches

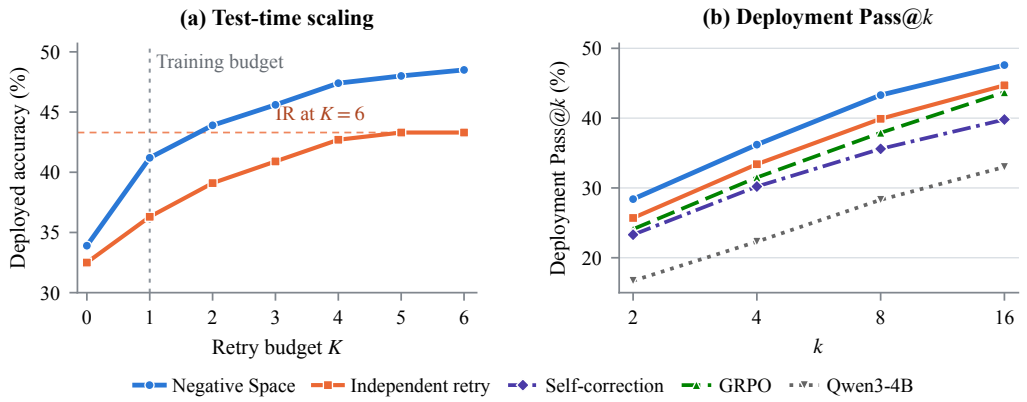


Figure 3: **(a) Test-time scaling.** Deployed accuracy on LiveCodeBench versus retry budget K, comparing NEGATIVE SPACE with RL with independent retry, the strongest baseline in the main code results. Both are trained with $K = 1$; guides mark this budget and independent retry's accuracy at $K = 6$. **(b) Deployment Pass@k.** Pass@k on CodeContests over k independent deployment runs, each returning one response y^{out} with $K = 1$.

RL with independent retry using three, and two retries exceed its accuracy with six. The advantage is consistent across the evaluated range, with a margin of 4.7–5.2 percentage points after the first retry. This complements equation 9: additional unlearning steps reuse the same learned preconditioner $\mathbf{A}^\top\mathbf{A}$, and their empirical benefit extends beyond the training budget.

Deployment pass@k. Figure 3b reports deployment Pass@k on CodeContests: the probability that at least one of k independently returned responses y^{out} passes the scoring tests. NEGATIVE SPACE leads at all evaluated k, improving over RL with independent retry by 2.7–3.4 points, so its advantage persists when multiple complete deployment runs are used to increase solution coverage.

Repair rates and retry behavior. We compare each retry with its failed first attempt on LiveCodeBench (Figure 4). NEGATIVE SPACE makes the largest structural changes, measured by normalized tree edit distance, and achieves the highest repair rate, exceeding RL with independent retry by 4.0 percentage points while both produce retries close to the original response length. Self-correction makes smaller structural edits and produces shorter responses, but repairs fewer failures. These observations associate the gains of NEGATIVE SPACE with more successful revisions that depart further from the rejected program.

Component ablation. Table 3a ablates both components of NEGATIVE SPACE on LiveCodeBench. For the lower level, (a) setting $\mathbf{B} = 0$ disables test-time adaptation while retaining independent retry, reducing accuracy; (b) To test whether $\mathbf{B}$ encodes query-specific information rather than acting as a generic perturbation, we swap it with an update computed from another query's rejected response, matching its Frobenius norm at each layer. This swap performs even worse than $\mathbf{B} = 0$, supporting the importance of query-specific information in the local update.

For the upper level, replacing the learned $\mathbf{A}$ with a random factor hurts performance the most, demonstrating the value of the learned geometry. Together, these results highlight the complementary roles of query-specific updates and the geometry learned to make them useful.

Table 3: Component ablations and test-time cost on LiveCodeBench.

(a) Deployed accuracy (%).

Upper level ($\mathbf{A}$)		Lower level ($\mathbf{B}$)	
Learned	Random	Zero	Swapped
42.4	35.1	38.9	37.1

(b) Cost per output response.

Method	Time (s)	TFLOPs
RL with indep. retry	3.05 (1.00×)	78.3 (1.00×)
NEGATIVE SPACE	3.84 (1.26×)	145.6 (1.86×)

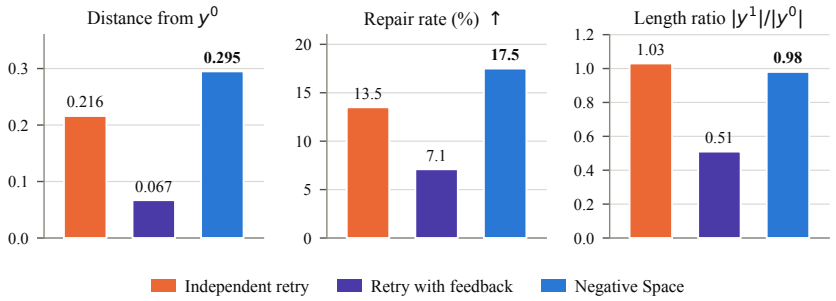


Figure 4: Retries compared with the failed first attempts they replace on LiveCodeBench. **Left:** normalized tree edit distance between the syntax trees of the failed attempt y^0 and the retry y^1; higher values mean the retry departs further from the failed program. **Middle:** repair rate, the fraction of rejected first attempts whose retry passes the hidden tests. **Right:** length ratio $|y^1|/|y^0|$; values near 1 indicate retries of typical length. NEGATIVE SPACE departs furthest from the failed program and repairs the most failures while preserving retry length.

Compute-aware comparison. We measure test-time cost on LiveCodeBench with two NVIDIA B200 GPUs. NEGATIVE SPACE uses $1.26\times$ the wall-clock time and $1.86\times$ the FLOPs of RL with independent retry per returned response (Table 3b). This adaptation overhead accompanies a smaller retry budget for comparable accuracy: NEGATIVE SPACE with two retries exceeds independent retry with six (Figure 3(a)). Together, these results characterize the trade-off between higher per-response cost and fewer sequential retries.

Hyperparameter analysis is provided in Appendix B.

6 RELATED WORK

Self-correction and test-time adaptation. SCoRe and related methods train policies to use previous attempts or feedback in subsequent reasoning (Kumar et al., 2024; Tang et al., 2025; Guo et al., 2026; Liu et al., 2026). TTRL estimates test-time rewards using majority voting (Zuo et al., 2025); tool- and verifier-guided methods also adapt parameters from inference-time signals (Liao et al., 2026; Moradi et al., 2025). ROSA uses interaction feedback for lightweight test-time policy adaptation (Wei et al., 2025). Thus feedback-conditioned parameter updates are not unique to ours. NEGATIVE SPACE instead learns from the failure: each rejection drives a negative-only local step whose effect is controlled by a geometry the upper level learns to improve the next retry.

Learning adaptation procedures. Gradient-based meta-learning trains parameters for subsequent adaptation (Finn et al., 2017). PERK uses a nested procedure to encode context in a LoRA adapter and learn to reason from it (Chen et al., 2026); Meta-TTL instead learns adaptation policies that revise an agent's prompts (Lou et al., 2026). NEGATIVE SPACE studies rejection-driven adaptation with an asymmetric factorization. Its shared factor is optimized through the adapted policy while the inner update is detached. This surrogate differs from exact meta-gradients through the adaptation.

Negative RLVR. RLVR methods such as GRPO (Shao et al., 2024) optimize group-based policy objectives in which incorrect responses receive negative advantages. Negative sample reinforcement (Zhu et al., 2025) shows that training only on incorrect responses can improve reasoning by suppressing them and redistributing probability toward other plausible candidates. These methods use negative samples during training to improve a single policy. NEGATIVE SPACE instead uses rejected attempts at test time: each rejection drives a query-specific unlearning step, and training learns a shared factor that shapes how such steps change the model.

7 CONCLUSION

We introduced NEGATIVE SPACE, a framework for test-time learning from rejected reasoning attempts. Through bilevel training, it learns an update geometry that makes local unlearning useful for subsequent retries. Experiments on code generation and mathematical reasoning show improved deployed accuracy over the strongest evaluated retry baselines. Further analysis shows that its advantage persists beyond the retry budget used during training.

Reproducibility statement

The method specifies the inner objective, detached outer surrogate, and retry weighting. Appendix A summarizes the reported code-training configuration. The main tables are based on individual trained checkpoints; reproducing a row requires its corresponding task split, feedback protocol, decoding configuration, and checkpoint.

Ethics statement

Optimization against incomplete tests or consensus can reinforce spurious solutions. Generated code is executed in an isolated evaluation environment, and private evaluation labels are excluded from test-time feedback. The experiments use existing code and mathematics benchmarks and do not involve human subjects.

AI use statement

Generative AI tools were used for initial outlining, literature organization, and language editing. The authors reviewed all AI-assisted text and take responsibility for the paper's claims, citations, code, and results.

References

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. MathArena: Evaluating LLMs on uncontaminated math competitions. *arXiv preprint arXiv:2505.23281*, 2025. URL https://arxiv.org/abs/2505.23281.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024. URL https://arxiv.org/abs/2407.21787.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/2460396f2d0d421885997dd1612ac56b-Abstract-Conference.html.

Zeming Chen, Angelika Romanou, Gail Weiss, and Antoine Bosselut. PERK: Long-context reasoning as parameter-efficient test-time learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=qxDTe8fIyA.

Yinlam Chow, Guy Tennenholtz, Izzeddin Gur, Vincent Zhuang, Bo Dai, Aviral Kumar, Rishabh Agarwal, Sridhar Thiagarajan, Craig Boutilier, and Aleksandra Faust. Inference-aware fine-tuning for Best-of-N sampling in large language models. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c40bed606c51c8e827c1ba75aa2da054-Abstract-Conference.html.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, 2017. URL https://proceedings.mlr.press/v70/finn17a.html.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 19034–19055, 2025. URL https://proceedings.mlr.press/v267/gehring25a.html.

Weiyang Guo, Zesheng Shi, Longhui Zhang, Zeen Zhu, Min Zhang, and Jing Li. Agent reinforcement learning via pivotal-aware self-feedback retry. *arXiv preprint arXiv:2607.03702*, 2026. URL https://arxiv.org/abs/2607.03702.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen 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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

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. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

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. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024. URL https://arxiv.org/abs/2409.12917.

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://doi.org/10.1126/science.abq1158.

Ruotong Liao, Nikolai Röhrich, Xiaohan Wang, Yuhui Zhang, Yasaman Samadzadeh, Volker Tresp, and Serena Yeung-Levy. Tool verification for test-time reinforcement learning. *arXiv preprint arXiv:2603.02203*, 2026. URL https://arxiv.org/abs/2603.02203.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

Yuanxin Liu, Ruida Zhou, Xinyan Zhao, Amr Sharaf, Hongzhou Lin, Arijit Biswas, Mohammad Ghavamzadeh, Zhaoran Wang, and Mingyi Hong. REVES: REvision and VErification–augmented training for test-time scaling. *arXiv preprint arXiv:2606.18910*, 2026. URL https://arxiv.org/abs/2606.18910.

Zhanzhi Lou, Hui Chen, Yibo Li, Qian Wang, and Bryan Hooi. Learning to learn-at-test-time: Language agents with learnable adaptation policies. *arXiv preprint arXiv:2604.00830*, 2026. URL https://arxiv.org/abs/2604.00830.

Mohammad Mahdi Moradi, Hossam Amer, Sudhir Mudur, Weiwei Zhang, Yang Liu, and Walid Ahmed. Continuous self-improvement of large language models by test-time training with verifier-driven sample selection. *arXiv preprint arXiv:2505.19475*, 2025. URL https://arxiv.org/abs/2505.19475.

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

Chenming Tang, Hsiu-Yuan Huang, Weijie Liu, Clive Bai, Saiyong Yang, and Yunfang Wu. Do not step into the same river twice: Learning to reason from trial and error. *arXiv preprint arXiv:2510.26109*, 2025. URL https://arxiv.org/abs/2510.26109.

Chenxing Wei, Hong Wang, Ying He, Fei Yu, and Yao Shu. Test-time policy adaptation for enhanced multi-turn interactions with LLMs. *arXiv preprint arXiv:2509.23166*, 2025. URL https://arxiv.org/abs/2509.23166.

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. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. SimpleRL-Zoo: Investigating and taming zero reinforcement learning for open base models in the wild. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=vSMCBUgrQj.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. The surprising effectiveness of negative reinforcement in LLM reasoning. *arXiv preprint arXiv:2506.01347*, 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, pp. 131459–131483, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/be690ea16f005c174f6c4102a5970e67-Abstract-Conference.html.

A IMPLEMENTATIONDETAILS

A.1 CODING SETUP

Table 4 records the code configuration used in the main experiment description.

Test separation. CodeContests public tests provide adaptation feedback. On training tasks, 15 additional private/generated tests provide the outer reward. On held-out tasks, the full additional test suite provides the evaluation score, with a reported median of 201 tests. These evaluation outcomes are not used as adaptation feedback. Public tests have a median count of two per training task and one per evaluation task.

Training schedule. The code model uses rank $r = 16$, scaling $\alpha = 16$, inner step size $\eta = 0.1$. GRPO uses learning rate 10^{-6}, 64 prompts per step, and $M = 8$ initial samples per prompt. Training runs on four B200 GPUs. The reported code training pool contains 7,054 rows.

Table 4: Reported configuration for the current CodeContests training runs.

Component	Value
LoRA	$r = 16$, $\alpha = 16$
Adapter targets	q, k, v, o, gate, up, down projections; every block
Inner step	$\eta = 0.1$;
learning rate	10^{-6}
Initial group / prompt batch	$M = 8$ / 64 prompts
Training hardware	$4\times$ B200

A.2 MATHEMATICS SETUP

Data and verifiers. We train on 8,520 problems of levels 3–5 of MATH (Hendrycks et al., 2021) for one epoch, at 64 prompts and $M = 8$ samples per step. Nothing in the deployed loop sees a reference answer: V draws N samples for the problem, takes the majority answer as a pseudo-label following Zuo et al. (2025), and rejects an attempt whose answer disagrees with it ($N = 8$ in training, $N = 16$ at evaluation.

Models and training. We employ one epoch at 64 prompts $\times$ 8 samples with a 1,024-token prompt budget and an 8,192-token response budget, sample at temperature 0.6 and $p = 0.95$, use policy learning rate 5×10^{-7}, and are scored at their final checkpoint. The three retry-loop rows adapt a rank-32 LoRA ($\alpha = 32$); Inner step is $\eta = 0.3$.

Evaluation. We use temperature 1.0, $p = 0.95$, an 8,192-token response budget and $K_{\max} = 1$: the deployed attempt is selected by V and scored by V_{h}.

B HYPERPARAMETER ANALYSIS

We examine the effects of the inner step size η and adapter rank d on CodeContests, varying one hyperparameter at a time around the default configuration. As shown in Table 5, accuracy is highest at $\eta = 0.1$ and decreases as the step size increases further. Increasing the rank from $d = 8$ to $d = 16$ improves accuracy, while doubling it to $d = 32$ yields only another 0.4 percentage points. The default rank of 16 therefore captures most of the observed benefit. All tested configurations match or exceed the strongest baseline on CodeContests, RL with independent retry (25.7%).

Table 5: Hyperparameter analysis on CodeContests, measured by deployed accuracy (%). Defaults are $\eta = 0.1$ and $d = 16$.

	Step size η				Rank d		
	0.03	0.1	0.3	1.0	8	16	32
Acc. (%)	29.6	31.1	27.6	25.7	28.6	31.1	31.5